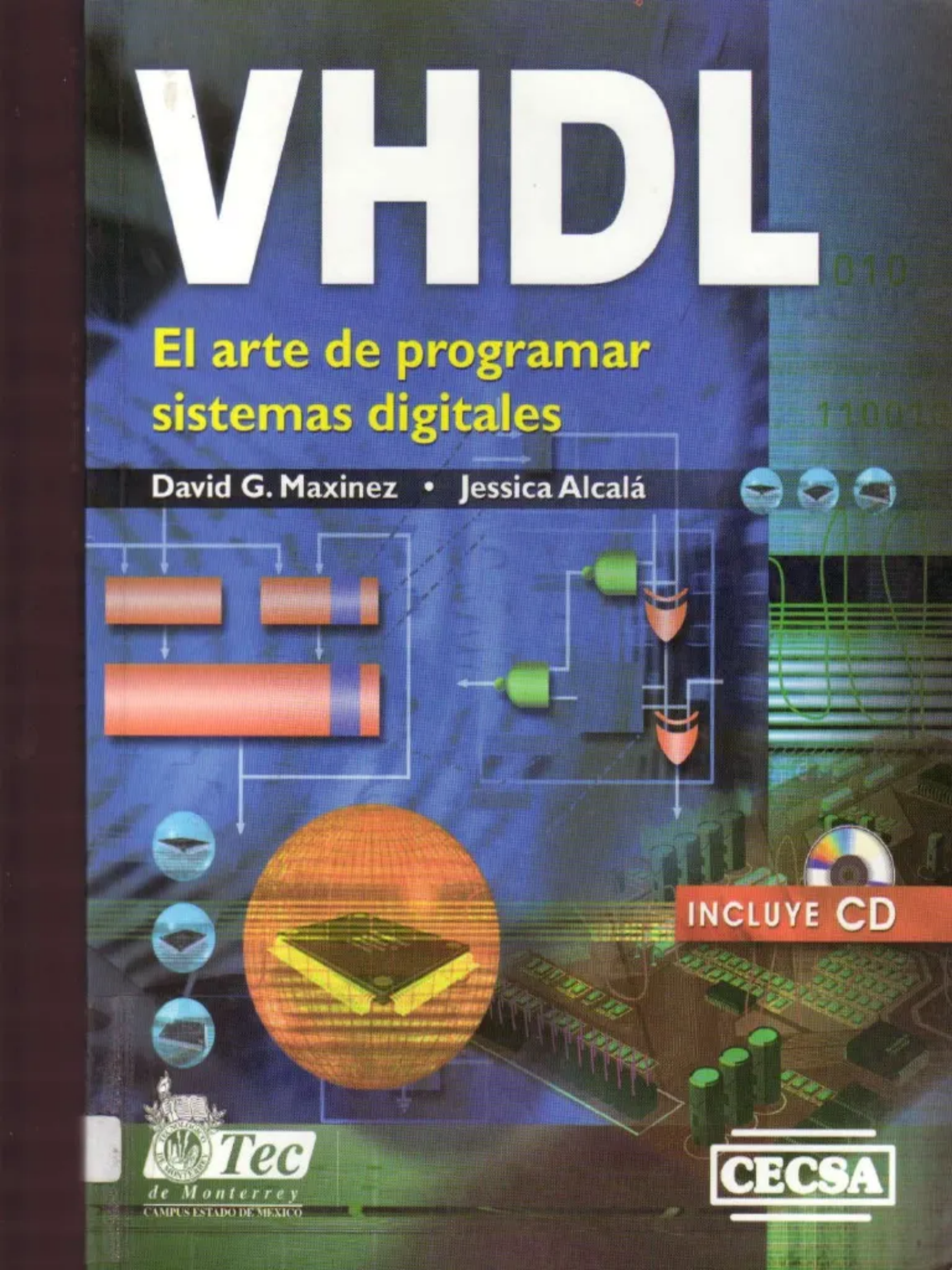
VHDL

El arte de programar sistemas digitales

David G. Maxinez • Jessica Alcalá



INCLUYE CD

Tec de Monterrey
CAMPUS ESTADO DE MÉXICO

CECSA

VHDL
El arte de programar sistemas digitales

David G. Maxinez
Jessica Alcalá Jara

Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Estado de México

PRIMERA EDICIÓN
MÉXICO, 2002

COMPAÑÍA EDITORIAL CONTINENTAL

Contenido

Acerca de los autores ix
Prólogo xi

1. Estado actual de la lógica programable 1
1.1 Dispositivos lógicos programables (PLD) 2
1.2 Dispositivos lógicos programables de alto nivel de integración 13
1.3 Ambiente de desarrollo de la lógica programable 18
1.4 Campos de aplicación de la lógica programable 23
1.5 La lógica programable y los lenguajes de descripción en hardware (HDL) . 25
1.6 Compañías de soporte en hardware y software 28
Ejercicios 33

2. VHDL: su organización y arquitectura 37
2.1 Unidades básicas de diseño 37
2.2 Entidad 38
2.3 Declaración de entidades 40
2.4 Diseño de entidades utilizando vectores 42
2.5 Arquitecturas (architecture) 46
Ejercicios 56

3. Diseño lógico combinacional mediante VHDL 61
3.1 Programación de estructuras básicas mediante declaraciones concurrentes . 61
3.2 Programación de estructuras básicas mediante declaraciones secuenciales . 69
Ejercicios 89

4. Diseño lógico secuencial con VHDL . . . 93
4-1 Diseño lógico secuencial . . . 93
4.2 Flip-Flops . . . 94
4.3 Registros . . . 98
4.4 Contadores . . . 101
4-5 Diseño de sistemas secuenciales síncronos . . . 105
Ejercicios . . . 113

5. Integración de entidades en VHDL . . . 123
5.1 Esquema básico de integración de entidades . . . 123
5.2 Integración de entidades básicas . . . 128
Ejercicios . . . 147

6. Diseño de controladores digitales mediante cartas ASM y VHDL . . . 153
6.1 El algoritmo de la máquina de estado (ASM) . . . 154
6.2 Estructura de una carta ASM . . . 156
6.3 Cartas ASM en comparación con las máquinas de estado . . . 159
6.4 Diseño de controladores mediante cartas ASM . . . 162
6.5 Diseño de cartas ASM mediante VHDL . . . 166
Ejercicios . . . 180

7. **Diseño jerárquico en VHDL . . . 197**
7.1 Metodología de diseño de estructuras jerárquicas . . . 198
7.2 Análisis del problema y descomposición en bloques individuales de la estructura global . . . 200
7.3 Diseño y programación de componentes o unidades del circuito . . . 201
7.4 Creación de un paquete de componentes . . . 206
7.5 Diseño del programa de alto nivel (Top Level) . . . 207
7.6 Creación de una librería en Warp . . . 208
Ejercicios . . . 225

8. Sistemas embebidos en VHDL . . . 229
8.1 Sistemas embebidos . . . 229
8.2 Diseño de un microprocesador . . . 237
8.3 Diseño jerárquico . . . 261
Ejercicios . . . 268

9. Redes neuronales artificiales y VHDL . . . 273
9.1 ¿Qué es una red neuronal artificial? . . . 275
9.2 Aprendizaje en las neuronas artificiales . . . 279
9.3 Aprendizaje por error mínimo . . . 291
9.4 Redes asociativas . . . 294
Ejercicios . . . 308

Apéndices

A. Herramientas de soporte para la programación en VHDL 311

B. Instalación del Software Warp 331

C. Identificadores, tipos y atributos 333

D. Hojas técnicas del CPLD CY7C372Í 343

E. Palabras reservadas en VHDL 347

F. Operadores definidos en VHDL según su orden de precedencia 349

Indice analítico 351

Acerca de los autores

David González Maxinez realizó sus estudios de licenciatura en la Universidad Nacional Autónoma de México, en la carrera de Ingeniero Mecánico Electricista con especialidad en el área de comunicaciones y electrónica. Posteriormente obtuvo el grado de maestro en ingeniería con "especialidad en electrónica" dentro de la división de estudios de posgrado de la Facultad de Ingeniería de la UNAM. Dentro del Instituto Tecnológico y de Estudios Superiores de Monterrey (ITESM), Campus Estado de México, cursó el Diplomado en Habilidades Docentes y posteriormente realizó el Diplomado en Microelectrónica en el ITESM, Campus Querétaro. Realizó sus estudios de Doctorado en el área de microelectrónica dentro de la Universidad Autónoma Metropolitana en convenio con la Universidad de TEXAS A&M en los Estados Unidos, desarrollando como tema de tesis doctoral: "El diseño de un microcontrolador NEUROFUZZY", el cual fue considerado de vanguardia en el ámbito de investigación nacional. Actualmente es profesor/investigador del Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Estado de México, así como de la Universidad Nacional Autónoma de México en el área de sistemas digitales. Combina esta actividad con el desarrollo de prototipos electrónicos aplicables a diversas áreas, mediante la integración en dispositivos lógicos programables.

Jessica Alcalá jara estudió Ingeniería en Computación en la Universidad Nacional Autónoma de México y es experta en el área de programación de sistemas digitales utilizando lenguajes de alto nivel. Dentro del ITESM, ha publicado el libro "Programación en VHDL". Actualmente es profesora/investigadora de la Universidad Nacional Autónoma de México, Campus Aragón, donde imparte las cátedras de organización de computadoras y microcomputadoras.

Los autores han participado juntos en congresos nacionales e internacionales en el área de lógica programable, sistemas difusos y microelectrónica.

Prólogo

Hoy en día, en nuestro ambiente familiar o de trabajo nos encontramos rodeados de sistemas electrónicos muy sofisticados. Teléfonos celulares, computadoras personales, televisores portátiles, equipos de sonido, dispositivos de comunicaciones y estaciones de juego interactivo, entre otros, no son más que algunos ejemplos del desarrollo tecnológico que ha cambiado nuestro estilo de vida haciéndolo cada vez más confortable. Todos estos sistemas tienen algo en común: su tamaño, de dimensiones tan pequeñas que resulta increíble pensar que sean igual o más potentes que los sistemas mucho más grandes que existieron hace algunos años.

Estos avances son posibles gracias al desarrollo de la microelectrónica, la cual ha permitido la miniaturización de los componentes para obtener así mayores beneficios de los chips (circuitos integrados) y para ampliar las posibilidades de aplicación.

La evolución en el desarrollo de los circuitos integrados se ha venido perfeccionando a través de los años. Primero, se desarrollaron los circuitos de baja escala de integración (SSI o Small Scale Integration), después los de mediana escala de integración (MSI o Médium Scale Integration) y posteriormente los de muy alta escala de integración (VLSI o Very Large Scale Integration) hasta llegar a los circuitos integrados de propósito específico (ASIC).

Actualmente, la gente encargada del desarrollo de nueva tecnología perfecciona el diseño de los circuitos integrados orientados a una aplicación y/o solución específica: los ASIC, logrando dispositivos muy potentes y que ocupan un mínimo de espacio. La optimización en el diseño de estos chips tiene dos tendencias en su conceptualización.

La primera tendencia es la técnica de *full custom design* (Diseño totalmente a la medida), la cual consiste en desarrollar un circuito para una aplicación específica mediante la integración de transistor por transistor. En su fabricación se siguen los pasos tradicionales de diseño: preparación de la oblea o base, el crecimiento epitaxial, la difusión de impurezas, la implantación de iones, la oxidación, la fotolitografía, la metalización y la limpieza química.

La segunda tendencia en el diseño de los ASIC proviene de una innovadora propuesta, que sugiere la utilización de celdas programables preestablecidas e insertadas dentro de un circuito integrado. Con base en esta idea surgió la familia de dispositivos lógicos programables (los PLD), cuyo nivel de densidad de integración ha venido evolucionando a través del tiempo. Iniciaron con los PAL (Arreglos Lógicos programables) hasta llegar al uso de los CPLD (Dispositivos Lógicos Programables Complejos) y los FPGA (Arreglos de Compuertas Programables en Campo), los cuales dada su conectividad interna sobre cada una de sus celdas han hecho posible el desarrollo de circuitos integrados de aplicación específica de una forma mucho más fácil y económica, para beneficio de los ingenieros encargados de integrar sistemas.

El contenido de este libro se encuentra orientado hacia este tipo de diseño. Nuestro objetivo es brindar al lector la oportunidad de comprender, manejar y aplicar el lenguaje de programación más poderoso para este tipo de aplicaciones: VHDL.

El lenguaje de descripción en hardware VHDL es considerado como la máxima herramienta de diseño por las industrias y universidades de todo el mundo, pues proporciona a los usuarios muchas ventajas en la planeación y diseño de los sistemas electrónicos digitales.

Esta obra ha sido preparada especialmente para aquellos estudiantes e ingenieros que desean introducirse en el manejo de este lenguaje de programación. El libro se encuentra redactado de una manera muy clara y accesible para guiar al lector paso por paso en los primeros cuatro capítulos. Para aquellas personas que ya tengan conocimientos sobre el tema, hemos incorporado una serie de libros recomendados con mayor profundidad y complejidad.

Uno de los objetivos del libro es proporcionar al estudiante y al ingeniero electrónico o de computación una forma fácil y práctica de integrar aplicaciones digitales utilizando el lenguaje de descripción en hardware VHDL. También esperamos motivar al lector para que comience el desarrollo e integración de sistemas electrónicos a través este lenguaje, con la visión y oportunidad de crecer como microempresario en el desarrollo de sistemas miniaturizados, los cuales pueden ser fácilmente comercializados, y generar así fuentes de empleo en beneficio de la sociedad.

Este libro es recomendable para un curso introductorio de VHDL, tanto a nivel técnico como a nivel universitario dado que para interpretar y entender los ejercicios sobre los que se realiza nuestra explicación sólo se requiere como antecedente un curso básico de diseño lógico que involucre el conocimientos

de los temas de compuertas lógicas, minimización de funciones booleanas, circuitos combinacionales y circuitos secuenciales.

Para nosotros es importante que los conocimientos que se adquieran en la lectura de este libro puedan ser trasladados de manera inmediata hacia la práctica o aplicación. Por ello, hemos incluido algunos datos de orientación para que el lector sepa cómo conseguir las herramientas de diseño y dónde conseguir de manera oportuna los circuitos PLD para integrar sus aplicaciones (véase el Apéndice A). También hemos incluido demostraciones del programa WARP en el CD que acompaña al libro.

Organización del libro

En resumen, esta obra está estructurada en nueve capítulos y cinco apéndices. En el capítulo uno se describe de forma cualitativa el estado actual de la lógica programable: sus antecedentes, ventajas, desventajas y perspectivas, además proporciona la información referente a las compañías que brindan el soporte tanto en software como hardware. Este capítulo es completamente informativo y su finalidad consiste en familiarizar y adentrar al lector en esta área de desarrollo.

En el capítulo dos *(VHDL: su organización y arquitectura*) se presenta la estructura básica del lenguaje de descripción en hardware VHDL y se analizan los módulos básicos de diseño: sus palabras reservadas, los tipos de datos, así como el manejo de las diferentes arquitecturas o estilos de diseño empleados en el desarrollo de un programa.

En el capítulo tres (*Diseño lógico combinacional mediante VHDL*) se describe la forma de emplear declaraciones concurrentes y secuenciales dentro de un programa mediante la solución de circuitos combinacionales individuales tales como los multiplexores, los decodificadores, los codificadores y sumadores, entre otros. Es importante resaltar que en este capítulo y en los subsecuentes no se presenta la solución óptima para un problema dado. Por el contrario, cada problema se aborda de diferente manera con el único fin de presentar al lector el uso cada vez mayor de nuevas palabras reservadas por VHDL en el entendido que la mejor solución de un problema se da por sí solo y llega cuando el lector conoce más y más formas de programar.

En el capítulo cuatro (*Diseño lógico secuencial con VHDL)* se realiza una síntesis de diseño de los principales circuitos secuenciales: flip-flop, registros de corrimiento, contadores, manejo de diagramas de estado etcétera. Todos ellos se analizan por separado haciendo énfasis en las instrucciones encargadas de sincronizar los eventos que se desarrollan en estas entidades de diseño.

El capítulo cinco (*Integración de entidades en VHDL)* detalla la forma en que se unen diferentes bloques lógicos, es decir, se describe cómo se integran dentro de una sola entidad varios circuitos, sean combinacionales y/o se-

cuenciales, manejados en su programación de manera individual o a través exclusivamente de la relación entrada/-salida.

En el capítulo seis *(Diseño de controladores digitales mediante cartas* ASM *y* VHDL) se describe la programación de algoritmos digitales (controladores), mostrando al lector la forma de como brindar soluciones a un problema dado a través del desarrollo y conceptualización de un algoritmo, cuya descripción se realiza a través de la carta ASM. En este capítulo el lector echa mano de todos los conocimientos adquiridos en los capítulos previos, sorprendiéndose de la facilidad con la cual se programan sistemas digitales complejos.

En el capítulo siete *(Diseño jerárquico en* VHDL) se presenta la forma de jerarquizar o dividir un problema en pequeños subsistemas que pueden analizarse y simularse de manera independiente para posteriormente entrelazarlos mediante un programa de alto nivel denominado Top Level. Esta forma de programación es muy utilizada en VHDL ya que permite crear nuevas librerías de trabajo, que el diseñador puede almacenar para posteriormente incluir en nuevos desarrollos.

En el capítulo ocho *(Sistemas embebidos en VHDL*) se incluye la parte teórica de los sistemas embebidos así como su ciclo de desarrollo mediante el diseño de un microprocesador, describiendo en detalle cada uno de sus módulos y la forma de programarlos mediante el diseño jerárquico o Top Level.

En el capítulo nueve dejamos el contexto de programación básica y abordamos la investigación realizada sobre la programación en VHDL de las redes neuronales artificiales. Esto tiene por objetivo bindar al lector una breve introducción a un nuevo campo de desarrollo e investigación, debido a que actualmente es posible desarrollar sistemas inteligentes a nivel de hardware e integrarlos en un circuito integrado, ya sea un CPLD o un FPGA.

Finalmente, los apéndices del libro se encuentran estructurados de la siguiente manera: en el apéndice A *(Herramientas de soporte para la programación en VHDL)* se describe el uso del software WarpR4 de Cypress Semiconductor, utilizado en la simulación de todos los ejercicios y problemas de este libro, además se incluyen los datos del distribuidor del software y de los circuitos integrados. El apéndice B contiene la información sobre cómo instalar el software en las diferentes plataformas tecnológicas. El apéndice C proporciona los principales identificadores, tipos y atributos que se manejan en VHDL, así como la sintaxis utilizada en cada declaración. En el apéndice D presentamos los datos técnicos del circuito CPLD CY372I-66JC de Cypress Semiconductor, en el cual fueron grabados los diseños presentados en este libro. En el apéndice E incluye una lista de palabras reservadas por VHDL con el fin de que el lector pueda identificarlas fácilmente y, por último, en el apéndice F se incluye una lista de operadores definidos en VHDL según su orden de precedencia.

Reconocimientos

Quisiéramos agradecerle a todos nuestros estudiantes tanto del Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Estado de México, como de la Universidad Nacional Autónoma de México, Campus Aragón, por su valiosa colaboración en la simulación e implementación de todos y cada uno de los ejercicios y problemas encontrados en este libro. En especial a Alejandra Abad Rodríguez, por su incomparable ayuda en esa etapa. De igual manera, deseamos agradecerle al Dr. Luis Niño de Rivera por sus comentarios y sugerencias como revisor técnico, al M. en C. Jorge Ramírez Landa, profesor del ITESM-CEM, y a la Ing. Ivette Cruz Felipe de la UNAM por su valiosa participación académica en el desarrollo de este libro. Finalmente, un agradecimiento a nuestra editora, Elisa Pecina, y a todo su equipo de trabajo siempre incansables en su afán de obtener ios mejores resultados y a quienes debemos la calidad de este trabajo.

David G. Maxinez
Jessica Alcalá Jara

Capítulo 1

Estado actual de la lógica programable

Introducción

En la actualidad el nivel de integración alcanzado con el desarrollo de la microelectrónica ha hecho posible desarrollar sistemas completos dentro de un solo circuito integrado SOC (*System On* Chip), con lo cual se han mejorado de manera notoria características como velocidad, confiabilidad, consumo de potencia y sobre todo el área de diseño. Esta última característica nos ha permitido observar día a día cómo los sistemas de uso industrial, militar y de consumo han minimizado el tamaño de sus desarrollos; por ejemplo, los teléfonos celulares, computadoras personales, calculadoras de bolsillo, agendas electrónicas, relojes digitales, sistemas de audio, sistemas de telecomunicaciones, etc., no son más que aplicaciones típicas que muestran la evolución de los circuitos integrados también conocidos como chips.

El proceso de miniaturización de los sistemas electrónicos comenzó con la interconexión de elementos discretos como resistencias, capacitores, resistores, bobinas, etc., todos colocados en un chasis reducido y una escasa separación entre ellos. Posteriormente se diseñaron y construyeron los primeros circuitos impresos —aún vigentes—, que relacionan e interconectan los elementos mencionados a través de cintas delgadas de cobre adheridas a un soporte aislante (por lo general baquelita) que permite el montaje de estos elementos [1].

Más tarde, el desarrollo del transistor de difusión planar, construido durante 1947 y 1948, permitió en 1960 la fabricación del primer circuito integrado monolítico. Este integra cientos de transistores, resistencias, diodos y capacitores, todos fabricados sobre una pastilla de silicio. Como es del conocimiento general, el término monolítico se deriva de las raíces griegas

"mono" y "lithos" que significan uno y piedra, respectivamente; por tanto, dentro de la tecnología de los circuitos integrados un circuito monolítico está construido sobre una piedra única o cristal de silicio que contiene tanto elementos activos (transistores, diodos), como elementos pasivos (resistencias, capacitores), y las conexiones entre ellos [1].

La fabricación de los circuitos monolíticos se basa en los principios de materiales, procesos y diseño que constituyen la tecnología altamente desarrollada de los transistores y diodos individuales. Dicha fabricación incluye la preparación de la oblea o base, el crecimiento epitaxial, la difusión de impurezas, la implantación de iones, la oxidación, la fotolitografía, la metalización y la limpieza química [1].

La integración de sistemas se ha ido superando a medida que surgen nuevas tecnologías de fabricación. Esto ha permitido obtener componentes estándares de mayor complejidad y que brindan mayores beneficios. Sin embargo, el desarrollo de nuevos productos requiere bastante tiempo, por lo cual sólo se emplea cuando se necesita un alto volumen de producción.

Una forma más rápida y directa de integrar aplicaciones es mediante la lógica programable, la cual permite independizar el proceso de fabricación del proceso de diseño fuera de la fábrica de semiconductores. Esta idea fue desarrollada por Hon y Sequin *[2]* y Conway y Mead [3] a finales de los años sesenta.

1.1 Dispositivos lógicos programables (PLD)

Los dispositivos lógicos programables (o PLD, por sus siglas en inglés) favorecen la integración de aplicaciones y desarrollos lógicos mediante el empaquetamiento de soluciones en un circuito integrado. El resultado es la reducción de espacio físico dentro de la aplicación; es decir, se trata de dispositivos fabricados y revisados que se pueden personalizar desde el exterior mediante diversas técnicas de programación. El diseño se basa en bibliotecas y mecanismos específicos de mapeado de funciones, mientras que su implementación tan sólo requiere una fase de programación del dispositivo que el diseñador suele realizar en unos segundos [4].

En la actualidad, el diseño de ASIC (circuitos integrados desarrollados para aplicaciones específicas) domina las tendencias en el desarrollo de aplicaciones a nivel de microelectrónica. Este diseño presenta varias opciones de desarrollo, como se observa en la tabla 1.1. A nivel de ASIC los desarrollos *full* y *semi custom* ofrecen grandes ventajas en sistemas que emplean circuitos diseñados para una aplicación en particular. Sin embargo, su diseño ahora sólo es adecuado en aplicaciones que requieren un alto volumen de producción; por ejemplo, sistemas de telefonía celular, computadoras portátiles, cámaras de video, etcétera.

Los FPGA (arreglos de compuertas programables en campo) y CPLD (dispositivos lógicos programables complejos) ofrecen las mismas ventajas de un ASIC, sólo que a un menor costo; es decir, el costo por desarrollar un ASIC es mucho más alto que el que precisaría un FPGA o un CPLD, con la ventaja de que ambos son circuitos reprogramables, en los cuales es posible modificar o borrar una función programada sin alterar el funcionamiento del circuito [4].

Categoría	Características
Diseño totalmente a la media (Full-Custom)	• Total libertad de diseño, pero el desarrollo requiere todas las etapas del proceso de fabricación: preparación de la oblea o base, crecimiento epitaxial, difusión de impurezas, implantación de iones, oxidación, fotolitografía, metalización y limpieza química [1]. Los riesgos y costos son muy elevados; sólo se justifican ante grandes volúmenes o proyectos con restricciones (área, velocidad, consumo de potencia, etcétera) [4].
Matrices de puertas predifundidas (Semicustom/gate arrays)	• Existe una estructura regular de dispositivos básicos (transistores) prefabricada que se puede personalizar mediante un conexionado específico que sólo necesita las últimas etapas del proceso tecnológico. • El diseño está limitado a las posibilidades de la estructura prefabricada y se realiza con base en una biblioteca de celdas precaracterizadas para cada familia de dispositivos [4].
Celdas estándares precaracterizadas (Semicustom/standard cells)	• No se trabaja con alguna estructura fija prefabricada en particular, pero sí con bibliotecas de celdas y módulos precaracterizados y específicos para cada tecnología. • Libertad de diseño (en función de las facilidades de la biblioteca); pero el desarrollo exige un proceso de fabricación completo [4].
Lógica programable (FPGA, CPLD).	• Se trata de dispositivos fabricados y revisados que se pueden personalizar desde el exterior mediante diversas técnicas de programación. • El diseño se basa en bibliotecas y mecanismos específicos de mapeado de funciones, mientras que su implementación tan sólo requiere una fase de programación del dispositivo, que por lo general realiza el diseñador en unos pocos segundos[4].

Tabla 1.1 Tecnologías de fabricación de circuitos integrados.

En la actualidad existe una gran variedad de dispositivos lógicos programables, los cuales se usan para reemplazar circuitos SSI (pequeña escala de integración), MSI (mediana escala de integración) e incluso circuitos VLSI (muy alta escala de integración), ya que ahorran espacio y reducen de manera significativa el número y el costo de los diseños. Estos dispositivos, llamados PLD (tabla 1.2), se clasifican por su arquitectura —la forma funcional en que se encuentran ordenados los elementos internos que proporcionan al dispositivo sus características.

Dispositivo	Descripción
PROM	Programmable Read-Only Memory: memoria programable de sólo lectura
PLA	Programmable Logic Array: arreglo lógico programable
PAL	Programmable Array Logic: lógica de arreglos programables
GAL	Generic Logic Array: arreglo lógico genérico
CPLD	Complex PLD: dispositivo lógico programable complejo
FPGA	Field Program Gate Array: arreglos de compuertas programmes en campo

Tabla 1.2 Dispositivos lógicos programables.

1.1.1 Estructura interna de un PLD

Los dispositivos PROM, PLA, PAL y GAL están formados por arreglos o matrices que pueden ser fijos o programables, mientras que los CPLD y FPGA se encuentran estructurados mediante bloques lógicos configurables y celdas lógicas de alta densidad, respectivamente.

La arquitectura básica de un PLD está formada por un arreglo de compuertas AND y OR conectadas a las entradas y salidas del dispositivo. La finalidad de cada una de ellas se describe a continuación.

a) Arreglo AND. Está formado por varias compuertas AND interconectadas a través de alambres, los cuales cuentan con un fusible en cada punto de intersección [Fig. 1.1a)]. En esencia, la programación del arreglo

consiste en fundir o apagar los fusibles para eliminar las variables que no serán utilizadas [Fig. 1.1b)]. Obsérvese cómo en cada entrada a las compuertas AND queda intacto el fusible que conecta la variable seleccionada con la entrada a la compuerta. En este caso, una vez que los fusibles se funden no pueden volver a programarse.

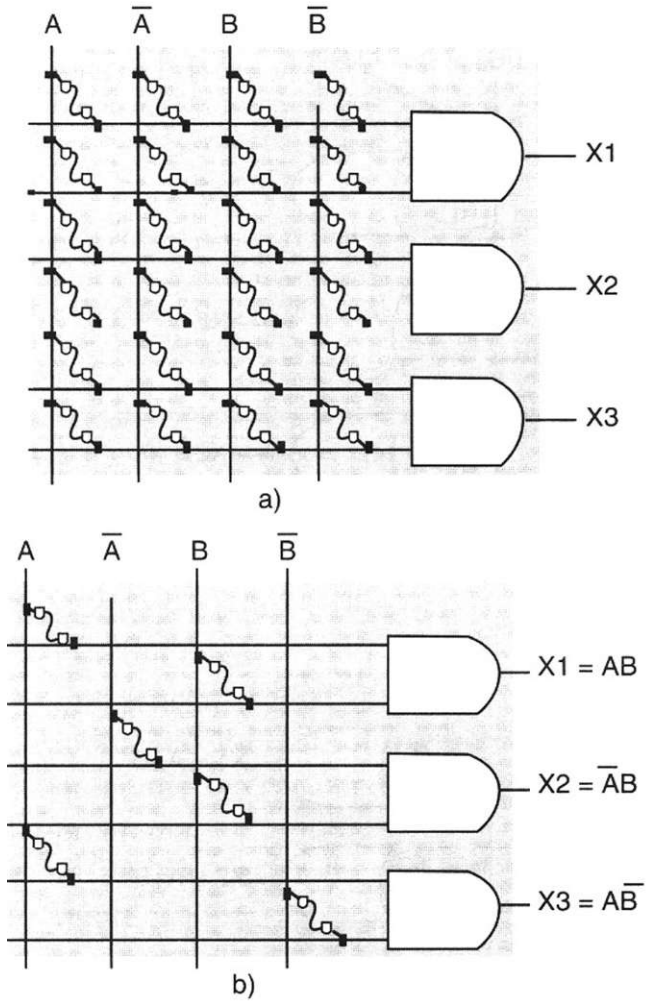


Figura 1.1 Arreglos AND: a) no programado y **b)** programado.

b) Arreglo OR. Está formado por un conjunto de compuertas OR conectadas a un arreglo programable, el cual contiene un fusible en cada punto de intersección. Este tipo de arreglo es similar al de compuertas AND explicado en el punto anterior, ya que de igual manera se programa fundiendo los fusibles para eliminar las variables no utilizadas. En la figura 1.2 se observa el arreglo OR programado y sin programar.

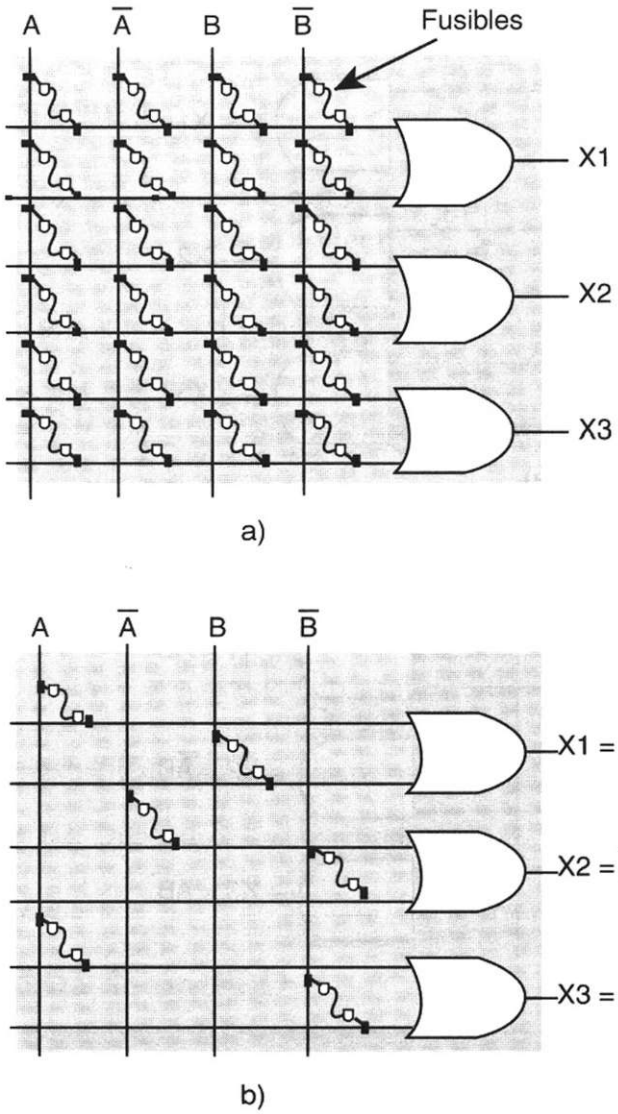


Figura 1.2 Estructura básica de PLD.

De acuerdo con lo anterior, observemos en la tabla 1.3 la estructura de los dispositivos lógicos programables básicos.

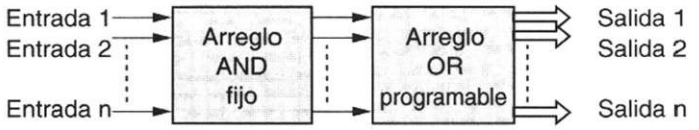
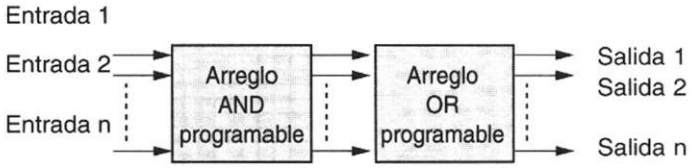
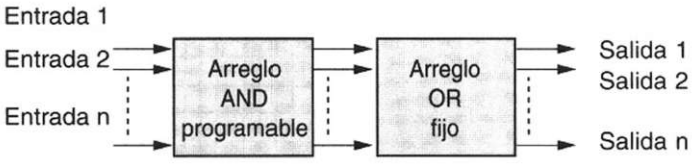
Dispositivo	Esquema básico
La memoria programable de sólo lectura (PROM) está formada por un arreglo no programable de compuertas AND conectadas como decodificador y un arreglo programable OR.	
El arreglo lógico programable (PLA) es un PLD formado por un arreglo AND y un arreglo OR, ambos programables.	
El arreglo lógico programable (PAL) se encuentra formado por los arreglos AND programable y OR fijo con lógica de salida.	

Tabla 1.3 Dispositivos lógicos programables.

- La PROM no se utiliza como un dispositivo lógico, sino como una memoria direccionable, debido a las limitaciones que presenta con las compuertas AND fijas.
- En esencia, el PLA se desarrolló para superar las limitaciones de la memoria PROM. Este dispositivo se llama también FPLA (arreglo lógico programable en campo), ya que es el usuario y no el fabricante quien lo programa.
- El PAL se desarrolló para superar algunas limitaciones del PLA, como retardos provocados por la implementación de fusibles adicionales, que resultan de la utilización de dos arreglos programables y de la complejidad del circuito. Un ejemplo típico de estos dispositivos es la familia PAL16R8, la cual fue desarrollada por la compañía AMD (Advanced Micro Devices) e incluye los dispositivos PAL16R4, PAL16R6, PAL16L8, PAL16R8, dispositivos programables por el usuario para reemplazar circuitos combinacionales y secuenciales SSI y MSI en un circuito.

En la figura 1.3 se muestra la arquitectura interna del PAL16L8. Como se puede observar, esta arquitectura está formada básicamente por circuitos combinacionales (compuertas AND, OR, buffers tri-estado e inversores), los cuales permiten la realización de funciones lógicas booleanas de la forma suma de productos (SOP). Cada término producto específico se programa en el arreglo AND, mientras que en el arreglo OR se realiza la suma lógica de los términos que se enviarán a las salidas del dispositivo.

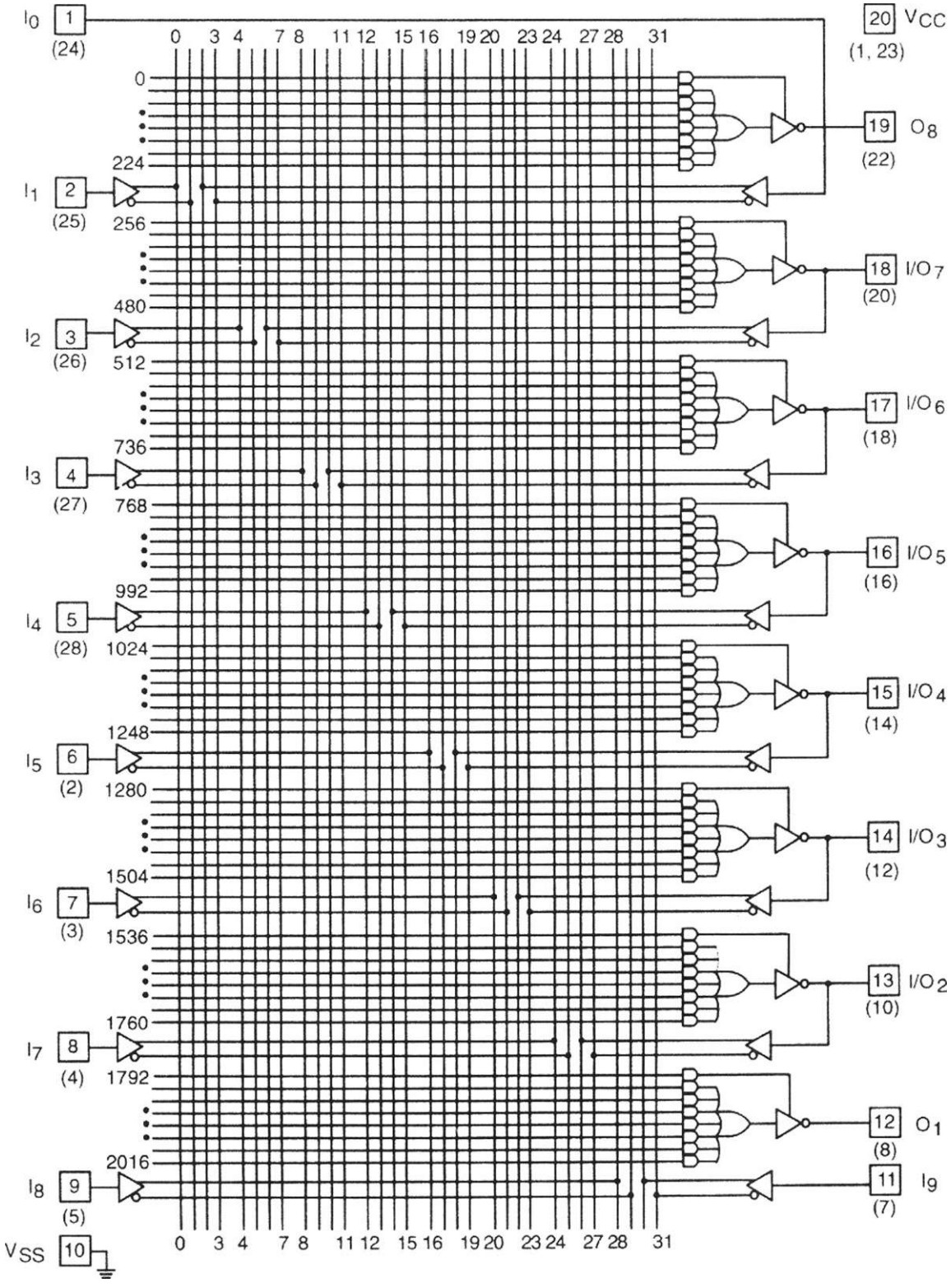


Figura 1.3 Arquitectura interna del PAL16L8.

1.1.2 Arreglo Lógico Genérico (GAL)

El arreglo lógico genérico (GAL) es similar al PAL, ya que se forma con arreglos AND programable y OR fijo, con una salida lógica programable. Las dos principales diferencias entre los dispositivos GAL y PAL radican en que el primero es reprogramable y contiene configuraciones de salida programables. Los dispositivos GAL se pueden programar una y otra vez, ya que usan la tecnología E^2 CMOS (Ellectrically Erasable CMOS: CMOS borrable eléctricamente), en lugar de tecnología bipolar y fusibles (Fig. 1.4).

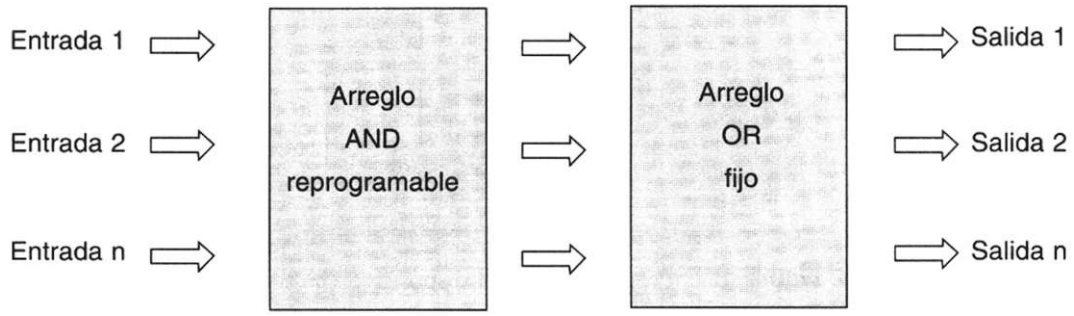


Figura 1.4 Diagrama de bloques del arreglo GAL.

Programación de un arreglo GAL

A diferencia de un PAL, el GAL está formado por celdas programables, las cuales se pueden reprogramar las veces que sea necesario. Como se observa en la figura 1.5, cada fila se conecta a una entrada de la compuerta AND y cada columna a una variable de entrada y sus complementos. Cuando se programa una celda, ésta se activa mediante la aplicación de cualquier combinación de las variables de entrada o sus complementos a la compuerta AND. Esto permite la implementación de cualquier función (producto de términos) requerida.

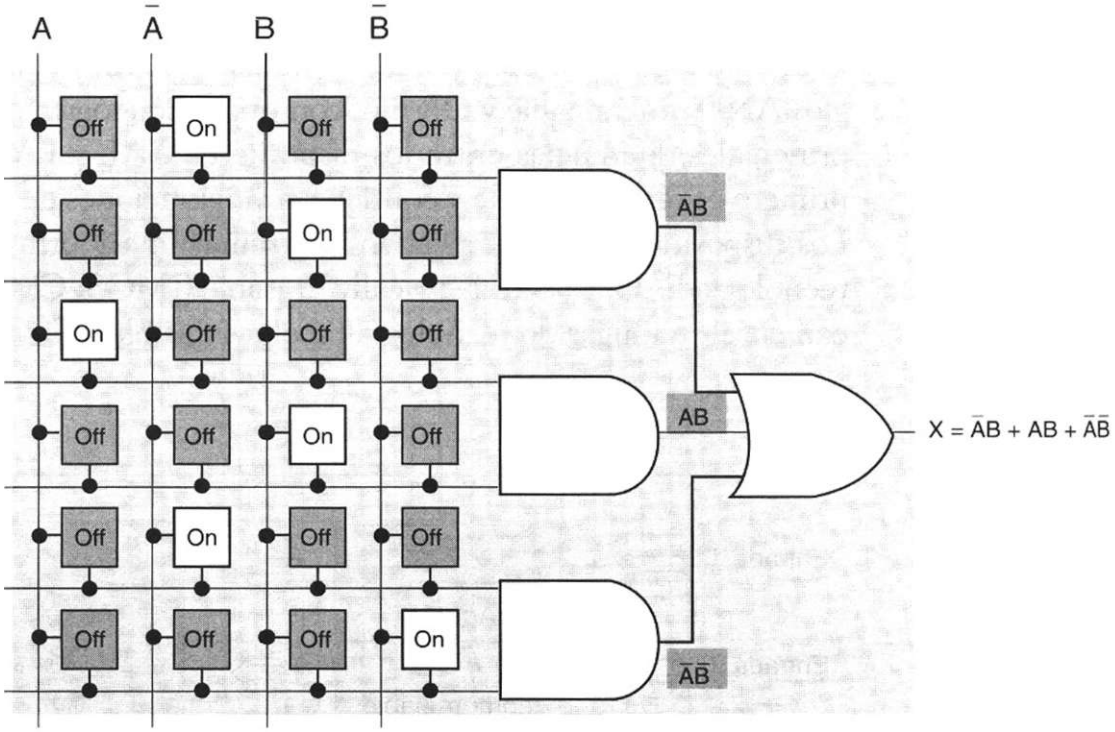


Figura 1.5 Programación del dispositivo GAL.

Arquitectura de un dispositivo GAL

Con el fin de esquematizar una arquitectura GAL, se toma como ejemplo el dispositivo GAL22V10 [Fig. 1.6a)]. Este circuito cuenta con 22 líneas de entrada y sus complementos, lo que da un total de 44 líneas de entrada a cada compuerta AND (estas entradas se encuentran representadas por las líneas verticales en el diagrama). La intersección que forman las líneas de entrada con los términos producto (líneas horizontales), representa cada una de las celdas que se pueden programar para conectar una variable de entrada (o su complemento) a una línea de término producto [Fig. 1.6b)], donde es posible apreciar la forma de obtener la suma de productos.

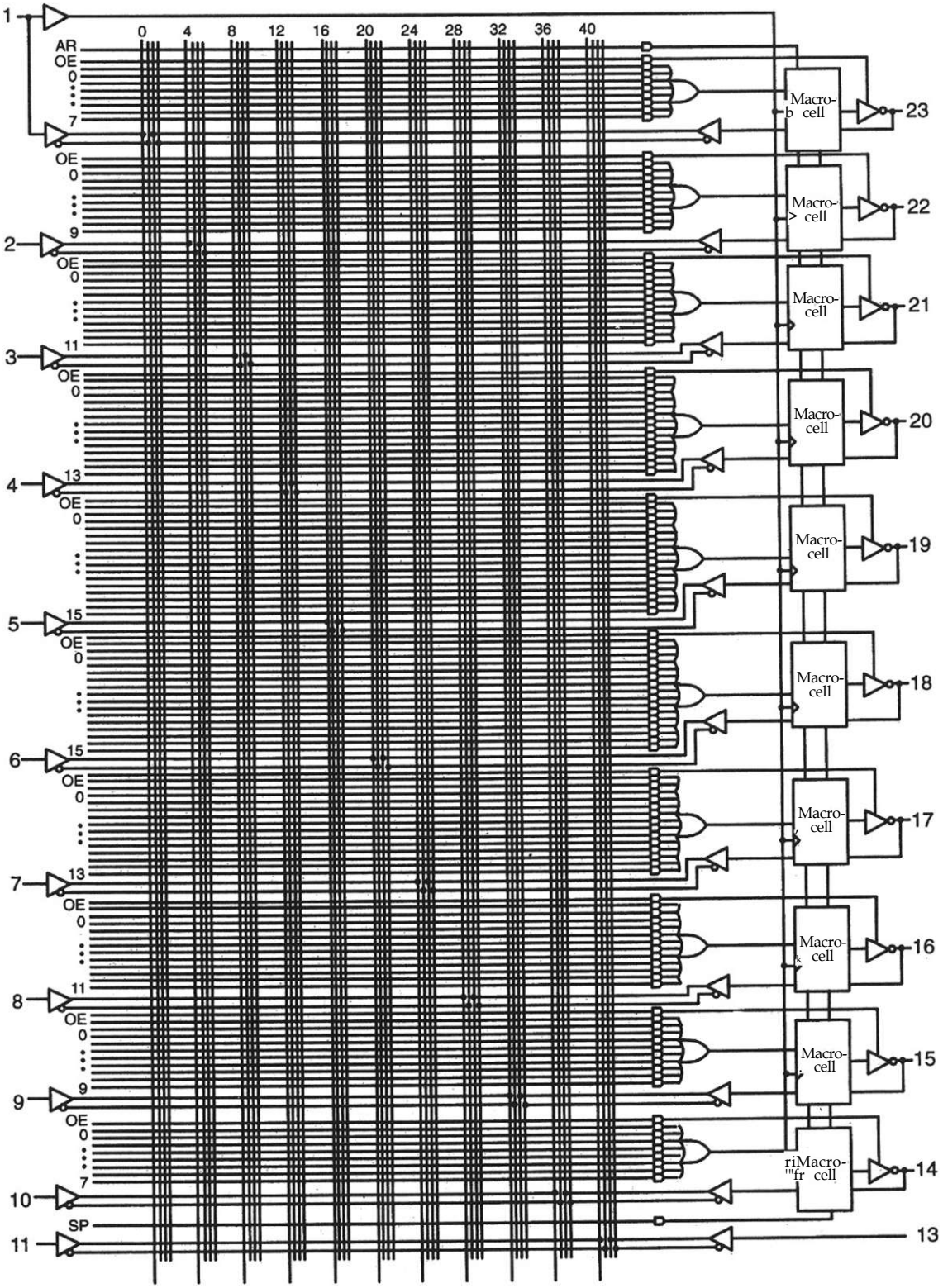


Figura 1.6a Arquitectura del GAL22V10.

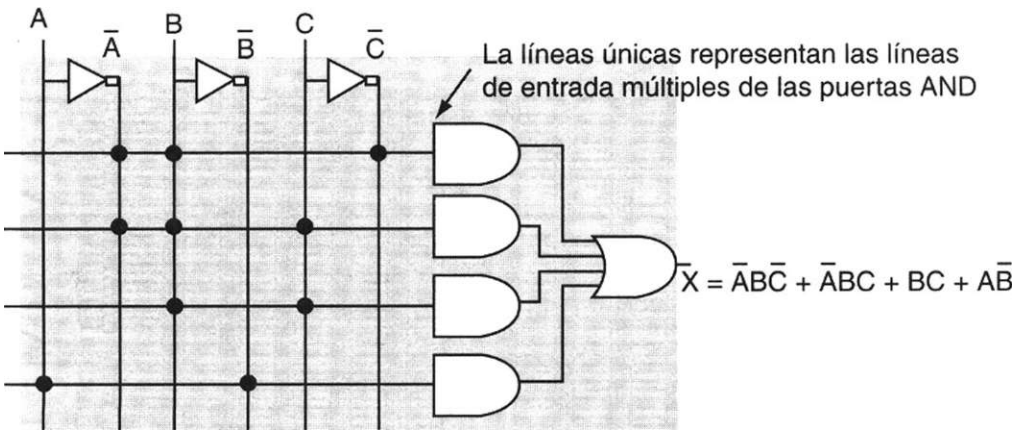


Figura 1.6b Realización de una suma de productos dentro de un GAL.

Macroceldas lógicas de salida. Una macrocelda lógica de salida (u OLMC, de output logic macrocell) está formada por circuitos lógicos que se pueden programar como lógica combinacional o secuencial [5]. Las configuraciones combinacionales se implementan por medio de programación, mientras que en las secuenciales la salida resulta de un flip-flop. En la figura 1.7 se observa la arquitectura de una macrocelda del dispositivo GAL22V10, la cual de manera general está formada por un flip-flop y dos multiplexores.

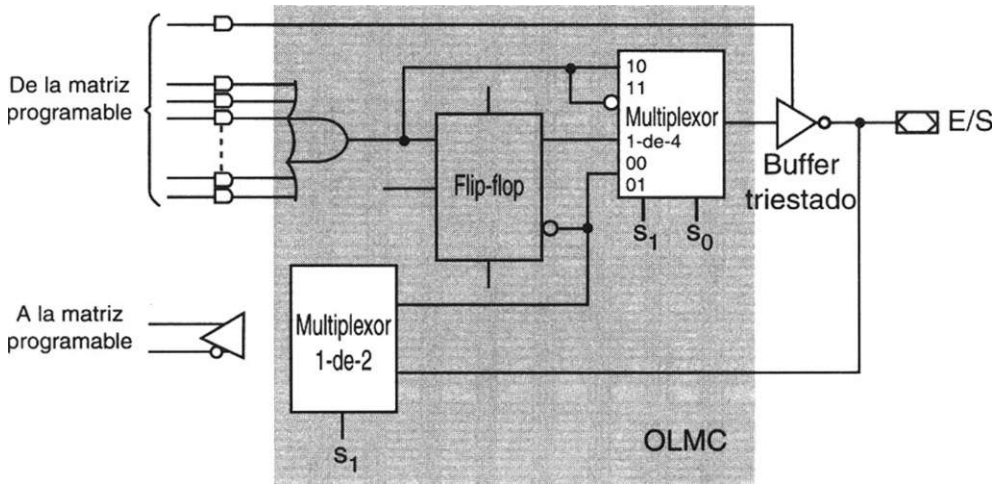


Figura 1.7 Arquitectura de una macrocelda OLMC 22VIO.

Puede haber de ocho a dieciséis entradas de las compuertas AND en la compuerta OR. Esto indica las operaciones producto que pueden efectuarse en cada macrocelda. El área punteada está formada por dos multiplexores y

un flip-flop; el multiplexor 1 de 4 conecta una de sus cuatro líneas de entrada al buffer triestado de salida, en función de las líneas de selección SO y SI. Por otro lado, el multiplexor de 1 de 2 conecta por medio del buffer la salida del flip-flop o la salida del buffer triestado al arreglo AND; esto se determina por medio de S1. Cada una de las líneas de selección se programa mediante un grupo de celdas especiales que se encuentran en el arreglo AND.

1.2 Dispositivos lógicos programables de alto nivel de integración

Los PLD de alto nivel de integración se crearon con el objeto de integrar mayor cantidad de dispositivos en un circuito (sistema en un chip SOC). Se caracterizan por la reducción de espacio y costo, además de ofrecer una mejora sustancial en el diseño de sistemas complejos, dado que incrementan la velocidad y las frecuencias de operación. Además, brindan a los diseñadores la oportunidad de enviar productos al mercado con más rapidez y les permiten realizar cambios en el diseño sin afectar la lógica, agregando periféricos de entrada/salida sin consumir una gran cantidad de tiempo, dado que los circuitos son reprogramables en el campo de trabajo.

1.2.1 Dispositivos lógicos programables complejos (CPLD)

Un circuito CPLD consiste en un arreglo de múltiples PLD agrupados como bloques en un chip. En algunas ocasiones estos dispositivos también se conocen como EPLD (Enhanced PLD: PLD mejorado), Super PAL, Mega PAL, [6j etc. Se califican como de alto nivel de integración, ya que tienen una gran capacidad equivalente a unos 50 PLD sencillos.

En su estructura básica, cada CPLD contiene múltiples bloques lógicos (similares al GAL22V10) conectados por medio de señales canalizadas desde la interconexión programable (PI). Esta unidad PI se encarga de interconectar los bloques lógicos y los bloques de entrada/salida del dispositivo sobre las redes apropiadas (Fig. 1.8).

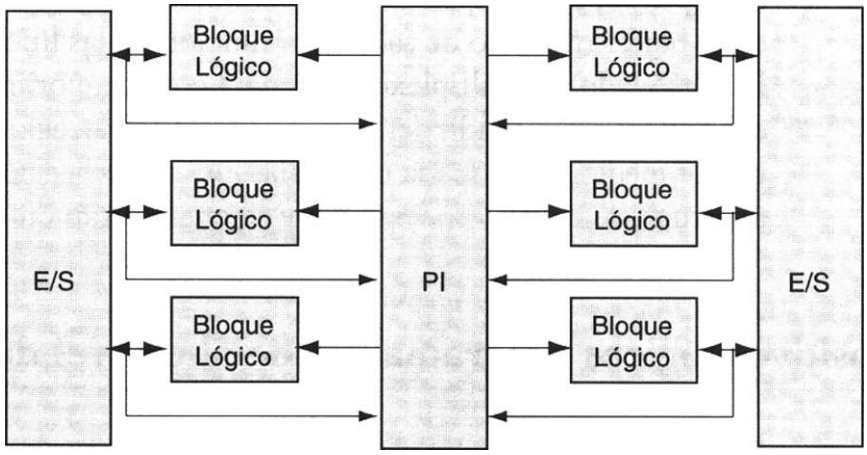


Figura 1.8 Arquitectura básica de un CPLD.

Los bloques lógicos, también conocidos como celdas generadoras de funciones, están formados por un arreglo de productos de términos que implementa los productos efectuados en las compuertas AND, un esquema de distribución de términos que permite crear las sumas de los productos provenientes del arreglo AND y por macroceldas similares a las incorporadas en la GAL22V10 (Fig. 1.9). En ocasiones las celdas de entrada/salida se consideran parte del bloque lógico, aunque la mayoría de los fabricantes coincide en que son externas. Cabe mencionar que el tamaño de los bloques lógicos es importante, ya que determina cuánta lógica se puede implementar dentro del CPLD; esto es, fija la capacidad del dispositivo.

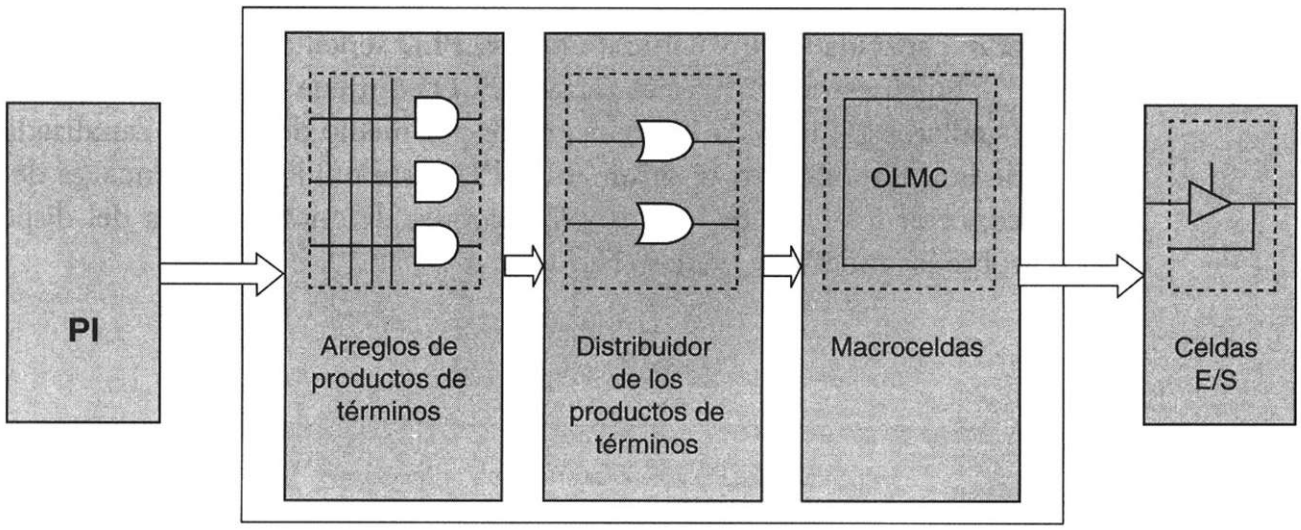


Figura 1.9 Bloque lógico programable.

a) Arreglos de productos de términos. Es la parte del CPLD que identifica el porcentaje de términos implementados por cada macrocelda y el número máximo de productos de términos por bloque lógico. Esto es similar al arreglo de compuertas AND programable de un dispositivo GAL22V10.

b) Esquema de distribución de términos. Es el mecanismo utilizado para distribuir los productos de términos a las macroceldas; esto se realiza mediante el arreglo programable de compuertas OR de un PLD. Los diferentes fabricantes de CPLD implementan la distribución de productos de términos con diferentes esquemas. Mientras algunos como la GAL22V10 usan un esquema de distribución variable (los cuales pueden variar en 8,10,12,14 o 16 productos por macrocelda), los CPLD como la familia MAX de Altera Corporation y Cypress Semiconductor, distribuyen cuatro productos de términos por macrocelda, además de utilizar "productos de términos expandidos", que se asignan a una o varias macroceldas.

c) Macroceldas. Una macrocelda de un CPLD está configurada internamente por flip-flops y un control de polaridad que habilita cada afirmación o negación de una expresión. Los CPLD suelen tener macroceldas de entrada/salida, de entrada y ocultas, mientras que los PLD sólo tienen macroceldas de entrada/salida.

La cantidad de macroceldas que contiene un CPLD es importante, debido a que cada uno de los bloques lógicos que conforman el dispositivo se expresan en términos del número de macroceldas que contiene. Por lo general, cada bloque lógico puede tener de cuatro a sesenta macroceldas; ahora bien, mientras mayor sea la cantidad, mayor será la complejidad de las funciones que se pueden implementar.

1.2.2 Arreglos de compuertas programables en campo (FPGA)

Los dispositivos FPGA se basan en lo que se conoce como arreglos de coni' puertas, los cuales consisten en la parte de la arquitectura que contiene tres elementos configurabies: bloques lógicos configurables (CLB), bloques de entrada y de salida (IOB) y canales de comunicación [7]. A diferencia de los CPLD, la densidad de los FPGA se establece en cantidades equivalentes a cierto número de compuertas.

Por adentro, un FPGA está formado por arreglos de bloques lógicos configurables (CLB), que se comunican entre ellos y con las terminales de entrada/salida (E/S) por medio de alambrados llamados canales de comunica' ción. Cada FPGA contiene una matriz de bloques lógicos idénticos, por lo general de forma cuadrada, conectados por medio de líneas metálicas que corren vertical y horizontalmente entre cada bloque (Fig. 1.10).

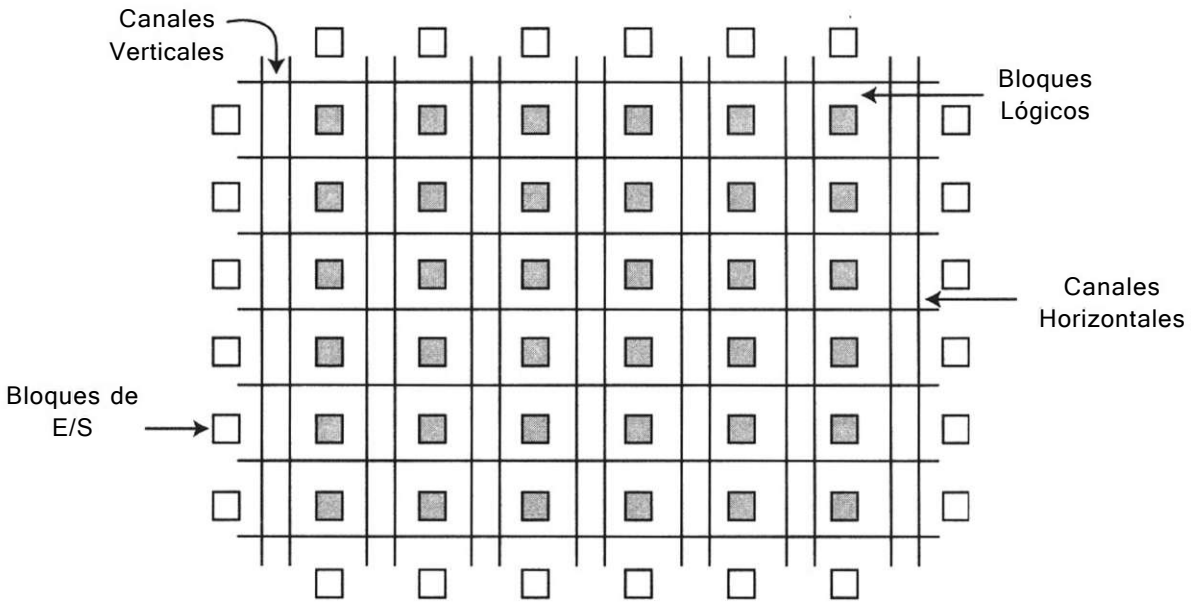


Figura 1.10 Arquitectura básica de un FPGA.

En la figura 1.11 se puede observar una arquitectura FPGA de la familia XC4000 de la compañía Xilinx. Este circuito muestra a detalle la configuración interna de cada uno de los componentes principales que conforman este dispositivo.

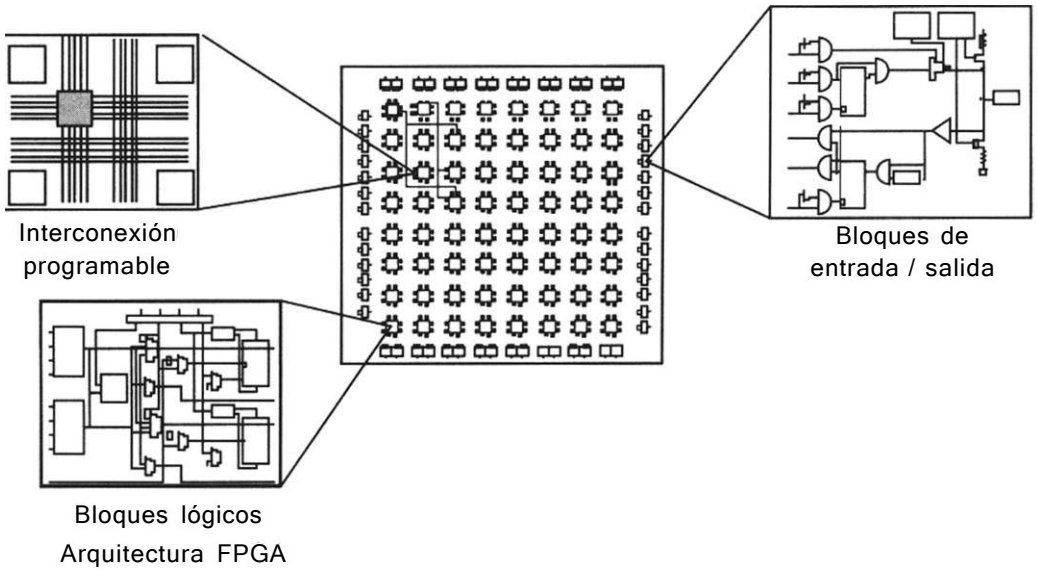


Figura 1.11 Arquitectura del FPGA XC4000 de Xilinx.

Los bloques lógicos (llamados también celdas generadoras de funciones) están configurados para procesar cualquier aplicación lógica. Estos bloques

tienen la característica de ser funcionalmente completos; es decir, permiten la implementación de cualquier función booleana representada en la forma de suma de productos. El diseño lógico se implementa mediante bloques conocidos como generadores de funciones o LUT (Look Up Table: tabla de búsqueda), los cuales permiten almacenar la lógica requerida, ya que cuentan con una pequeña memoria interna —por lo general de 16 bits— [6J. Cuando se aplica alguna combinación en las entradas de la LUT, el circuito la traduce en una dirección de memoria y envía fuera del bloque el dato almacenado en esa dirección. En la figura 1.12 se observan los tres LUT que contiene esta arquitectura, los cuales se encuentran etiquetados con las letras G, F y H.

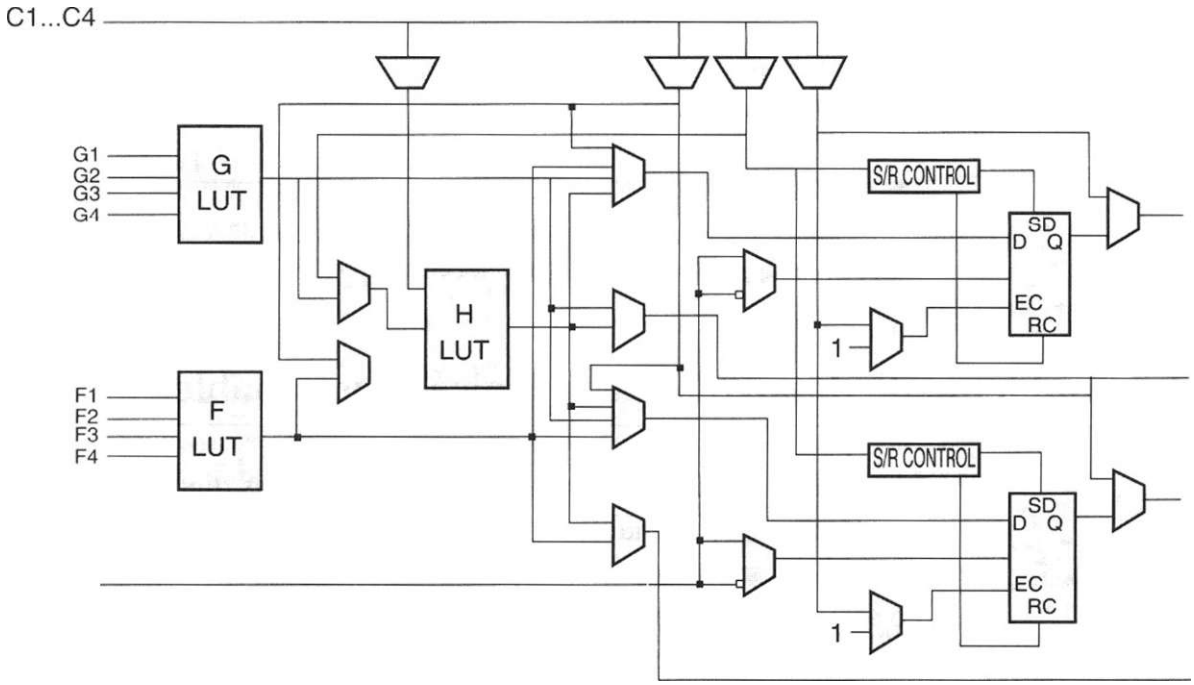


Figura 1.12 Arquitectura de un bloque lógico configurable FPGA.

En un dispositivo FPGA, los CLB están ordenados en arreglos de matrices programables (Programmable Switch Matrix o PSM), la matriz se encarga de dirigir las salidas de un bloque a otro. Las terminales de entrada y salida del FPGA pueden estar conectadas directamente al PSM o CLB, o se pueden conectar por medio de vías o canales de comunicación.

En algunas ocasiones se pueden confundir los dispositivos de FPGA y CPLD, ya que ambos utilizan bloques lógicos en su fabricación. La diferencia entre ellos radica en el número de flip-flops utilizados, mientras la arquitectura del FPGA es rica en registros. La CPLD mantiene una baja densidad. En la tabla 1.4 se presentan algunas otras diferencias entre una y otra arquitectura.

Características	CPLD	FPGA
Arquitectura	• Similar a un PLD • Más combinacional	• Similar a los arreglos de compuertas • Más registros + RAM
Densidad	• Baja a media	• Media a alta
Funcionalidad	• Trabajan a frecuencias superiores a 200 Mhz	• Depende de la aplicación (arriba de los 135Mhz)
Aplicaciones	• Contadores rápidos • Máquinas de estado • Lógica combinacional	• Excelentes en aplicaciones para arquitecturas de computadoras • Procesadores Digitales de Señales (DSP) • Diseños con registros

Tabla 1.4 Diferencias entre dispositivos lógicos programables complejos (CPLD) y los arreglos de compuertas programables en campo (FPGA).

1.3 Ambiente de desarrollo de la lógica programable

Una de las grandes ventajas al diseñar sistemas digitales mediante dispositivos lógicos programables radica en el bajo costo de los recursos requeridos para el desarrollo de estas aplicaciones. De manera general, el soporte básico se encuentra formado por una computadora personal, un grabador de dispositivos lógicos programables y el software de aplicación que soporta las diferentes familias de circuitos integrados PLD (Fig. 1.13).

Figura 1.13 Herramientas necesarias en la programación de PLD.

En la actualidad, diversos programas CAD (diseño asistido por computadora), como PALASM, OPAL, PLR ABEL, CUPL, entre otros, se encuentran disponibles para la programación de dispositivos lógicos (tabla 1.5).

Compilador lógico	Características
PALASM (PAL Assembler: ensamblador de PAL)	Creado por la compañía Advanced Micro Devices (AMD) Desarrollado únicamente para aplicaciones con dispositivos PAL Acepta el formato de ecuaciones booleanas Utiliza cualquier editor que grabe en formato ASCII
OPAL (Optimal PAL language: lenguaje de optimización para arreglos programables)	Desarrollado por National Semiconductors Se aplica en dispositivos PAL y GAL Formato para usar lenguaje de máquinas de estado, ecuaciones booleanas de distintos niveles, tablas de verdad, o cualquier combinación entre ellas. Disponible en versión estudiantil y profesional (OPAL Jr y OPAL Pro) Genera ecuaciones de diseño partiendo de una tabla de verdad
PLPL (Programable Logic Programming Language: lenguaje de programación de lógica programable)	Creado por AMD Introduce el concepto de jerarquías en sus diseños Formatos múltiples (ecuaciones booleanas, tablas de verdad, diagramas de estado y las combinaciones entre éstos) Aplicaciones en dispositivos PAL y GAL
ABEL (Advanced Boolean Expression Language: lenguaje avanzado de expresiones booleanas)	Creado por Data I/O Corporation Programa cualquier tipo de PLD (Versión 5.0) Proporciona tres diferentes formatos de entrada: ecuaciones booleanas, tablas de verdad y diagramas de estados. Es catalogado como un lenguaje avanzado HDL (lenguaje de descripción en hardware)
CUPL (Compiler Universal Programmable Logic: compilador universal de lógica programable)	Creado por AMD para desarrollo de diseños complejos Presenta una total independencia del dispositivo Programa cualquier tipo de PLD Facilita la generación de descripciones lógicas de alto nivel Al igual que ABEL, también es catalogado como HDL

Tabla 1.5 Descripción de compiladores lógicos para PLD.

Estos programas —conocidos también como compiladores lógicos— tienen una función en común: procesar y sintetizar el diseño lógico que se va a introducir en un PLD mediante un método específico (ecuaciones booleanas, diagramas de estado, tablas de verdad) [5].

Método tradicional de diseño con lógica programable

La manera tradicional de diseñar con lógica programable, parte de la representación esquemática del circuito que se requiere realizar y luego define la solución del sistema por el método adecuado (ecuaciones booleanas, tablas de verdad, diagramas de estado, etc.). Por ejemplo, en la figura 1.14 se observa un diagrama que representa a un circuito construido con compuertas lógicas AND y OR. En este caso se eligió el método de ecuaciones booleanas para representar su funcionamiento, aunque se pudo usar también una tabla de verdad.

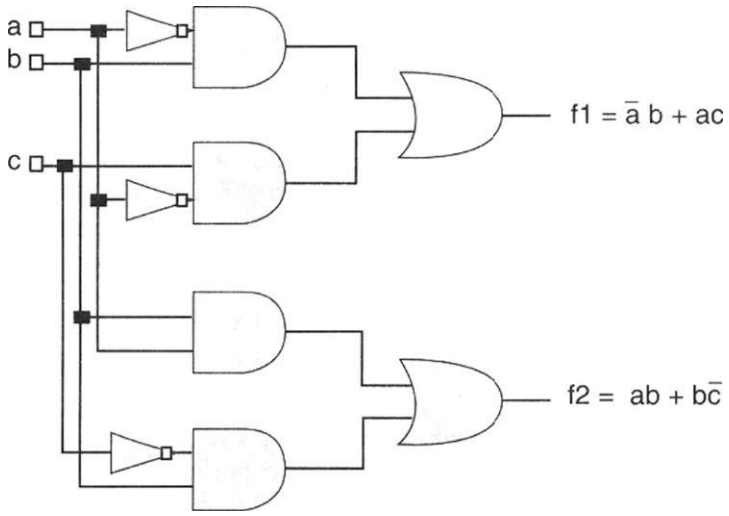


Figura 1.14 Obtención de las ecuaciones booleanas.

Como se puede observar, las ecuaciones que rigen el comportamiento del sistema se encuentran derivadas en función de las salidas fl y f2 del circuito. Una vez que se obtienen estas ecuaciones, el siguiente paso es introducir en la computadora el archivo fuente o de entrada; es decir, el programa que contiene los datos que permitirán al compilador sintetizar la lógica requerida. Típicamente se introduce alguna información preliminar que indique datos como el nombre del diseñador, la empresa, techa, nombre del diseño, etc. Luego se especifica el tipo de dispositivo PLD que se va a utilizar, la numeración de los pines de entrada y salida, y las variables del diseño. Por último, se define la función lógica en forma de ecuaciones booleanas o cualquier formato que acepte el compilador.

En la figura 1.15 se observa la pantalla principal del programa PALASM, en el cual se compilará el diseño de la figura 1.14 con el fin de ejemplificar la metodología que se debe seguir.

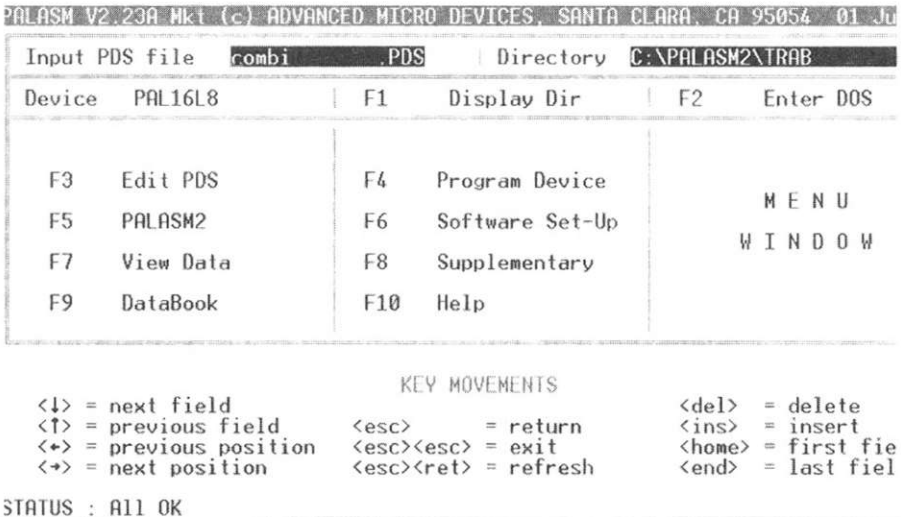


Figura 1.15 Pantalla principal de PALASM.

La forma de introducir el diseño se muestra en el listado 1. Nótese que las palabras reservadas por el compilador se representan con letras negritas.

```
TITLE           EJEMPLO
PATTERN         EJEMPLO.PDS
REVISION        1.0
AUTHOR          JESSICA
COMPANY         UNAM
DATE            00-00-00
CHIP            XX PAL16L8

; 1  2   3   4  5  6  7   8   9  10
NC  NC  NC  NC A  B  C  NC  NC  GND

; 1112  13  14  15  16  17  18  19   20
NC  NC  F1  F2  NC  NC  NC  NC  NC  1re c

EQUATIONS

F1 = /A* B + /A*C
F2 = A* B + B + /C

SIMULATION

TRACE_ON  A  B F1 F2

SETF /A /B /C
CHECK /F1 /F2
SETF /A /B  C
CHECK F1 /F2
SETF A /B C
CHECK F1 /F2

TRACE_OFF
```

Encabezado

Declaración de pines de entrada/salida

Ecuaciones del circuito

Simulación (condiciones e/s)

Listado 1.1 Archivo Fuente compilado en formato PALASM.

El siguiente paso consiste en la compilación del diseño, el cual radica básicamente en localizar los errores de sintaxis[1] o de otro tipo, cometidos durante la introducción de los datos en el archivo fuente. El compilador procesa y traduce el archivo fuente y minimiza las ecuaciones. En este paso, el diseño se ha simulado utilizando un conjunto de entradas y sus correspondientes valores de salida conocidos como vectores de prueba. Durante este proceso se comprueba que el diseño funcione correctamente antes de introducirlo al PLD. Si se detecta algún error en la simulación, se depura el diseño para corregir este defecto.

Una vez que el diseño no tiene errores, el compilador genera un archivo conocido como JEDEC (Joint Electronic Device Engineering Council)[2] o mapa de fusibles. Este archivo indica al grabador cuáles fusibles fundir y cuáles activar, para que luego se grabe el PLD (de acuerdo con el mapa de fusibles) en un grabador típico (Fig. 1.16).

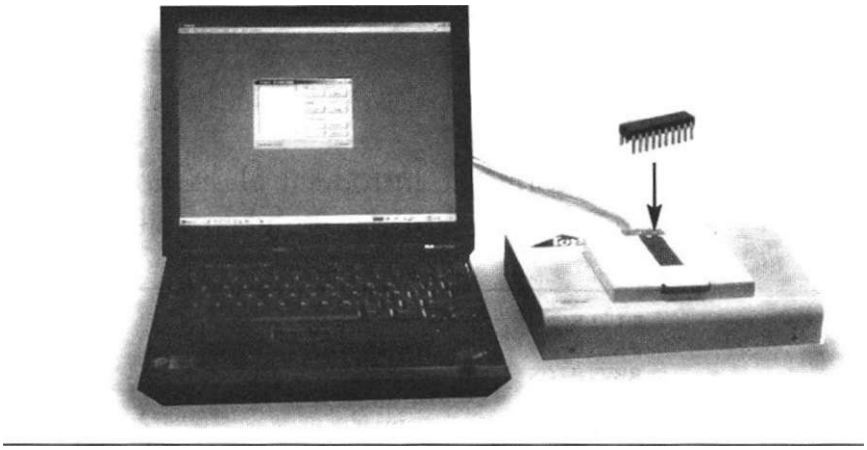


Figura 1.16 Implementación final del diseño en un PLD.

Como se puede observar, ciertos PLD (PROM, PAL, GAL) se programan empleando un grabador de dispositivos lógicos. Algunos otros PLD, como los CPLD y FPGA, presentan la característica de ser programables dentro del sistema (ISR In-System Programmable); es decir, no hay que introducirlos al grabador, ya que por medio de elementos auxiliares se pueden programar dentro de la tarjeta de circuito integrado.

Como se aprecia, el método de diseño con lógica programable reduce de manera considerable el tiempo de diseño y permite al diseñador mayor control de los errores que se pudieran presentar, ya que la corrección se realiza en el software y no en el diseño físico.

[1] La sintaxis se refiere al formato establecido y la-simbología utilizada para describir una categoría de funciones.
[2] Los archivos JEDEC están estabilizados para todos los compiladores lógicos existentes.

1.4 Campos de aplicación de la lógica programable

La lógica programable es una herramienta de diseño muy poderosa, que se aplica en el mundo industrial y en proyectos universitarios en todo el mundo. En la actualidad se usan desde los PLD más sencillos (como el GAL, PAL, PLA) como reemplazos de circuitos LSI y MSI, hasta los potentes CPLD y FPGA, que tienen aplicaciones en áreas como telecomunicaciones, computación, redes, medicina, procesamiento digital de señales, multiprocesamiento de datos, microondas, sistemas digitales, telefonía celular, filtros digitales programables, entre otros.

En general, los CPLD son recomendables en aplicaciones donde se requieren muchos ciclos de sumas de productos, ya que pueden introducirse en el dispositivo para ejecutarse al mismo tiempo, lo que conduce a pocos retrasos. En la actualidad, los CPLD son muy utilizados a nivel industrial, ya que resulta fácil convertir diseños compuestos por múltiples PLD sencillos en un circuito CPLD.

Por otro lado, los FPGA son recomendables en aplicaciones secuenciales que no suponen grandes cantidades de términos producto. Por ejemplo, los FPGA desarrollados por la compañía ATMEL ofrecen alta velocidad en cómputo intensivo, aplicaciones en procesadores digitales de señales (DSP) y en otras fases del diseño lógico, debido a la gran cantidad de registros con los que cuentan sus dispositivos (de 1024 a 6400). Esto los hace ideales para su uso en dichas áreas.

Desarrollos recientes

Existen desarrollos realizados por diversas compañías cuyo funcionamiento se basa en un PLD; por ejemplo, la figura 1.17 ilustra una tarjeta basada en un FPGA de la familia XC4000 de Xilinx Corporation. Este desarrollo permite el procesamiento de datos en paralelo a alta velocidad, lo que reduce los problemas de procesamiento de datos intensivo[3].

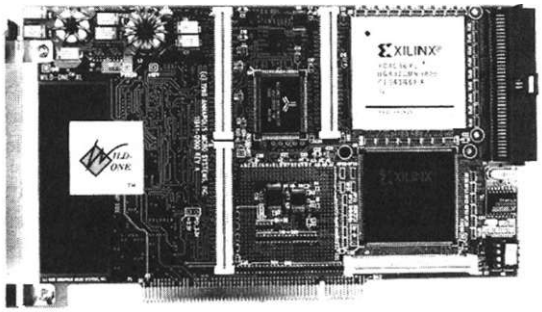


Figura 1.17 Sistema basado en un FPGA.

[1] Fuente de información: http://www.annapmicro.com

En la figura 1.18 se muestra otra aplicación realizada en un dispositivo CPLD de la familia FlexlOK de Altera Corporation (nivel de integración de 7000 compuertas). La función de esta tarjeta es permitir diversas aplicaciones en tiempo real, como el filtrado digital y muchas otras propias del campo del procesamiento digital de señales[4].

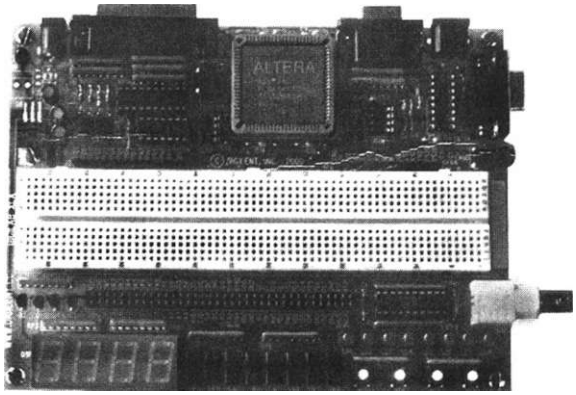


Figura 1.18 Ejemplo de un diseño lógico programable completo.

Como ya se mencionó, el campo de la lógica programable se ha extendido en la industria en los últimos años, ya que compañías de nivel internacional integran o desarrollan lógica programable en sus diseños (vea la tabla 1.6).

Compañía	Productos desarrollados con lógica programable
Andraka Consulting Group http://users.ids.net/~randraka/Inc	Procesadores digitales de señales (DSP) Comunicaciones digitales Procesadores de audio y video
Code Logic http://home.intekom.com/codelogic/	Lógica configurable Control embebido
Boton Line http://www.bltinc.com/	Modems de alta velocidad Audio, video, adquisición de datos y procesamiento de señales en general Aplicaciones militares: Criptografía, seguridad en comunicaciones, proyectos espaciales.
Comit's Services http://www.comit.com/	Redes: aplicaciones en protocolos TCP/IP Multimedia: Compresión de Audio/Video Aplicaciones en tiempo real
New Horizonts GU http://www.netcomuk.co.uk/~newhoriz/index.html	Digitalizadores, Cámaras de Video (120Mbytes/sec) Video en tiempo real Puertos paralelos de comunicaciones para PC
Design Service Segments http://www.smartech.fi/	Diseño de microprocesadores complejos Dispositivos para telecomunicaciones, DSP Aplicaciones en diseños para control industrial

Tabla 1.6 Compañías que incorporan lógica programable en sus diseños.

[4] Fuente de información: http://www.fgpa.com

1.5 La lógica programable y los lenguajes de descripción en hardware (HDL)

Como consecuencia de la creciente necesidad de integrar un mayor número de dispositivos en un solo circuito integrado, se desarrollaron nuevas herramientas de diseño que auxilian al ingeniero a integrar sistemas de mayor complejidad. Esto permitió que en la década de los cincuenta aparecieran los lenguajes de descripción en hardware (HDL) como una opción de diseño para el desarrollo de sistemas electrónicos elaborados. Estos lenguajes alcanzaron mayor desarrollo durante los años setenta, lapso en que se desarrollaron varios de ellos como IDL de IBM, TI-HDL de Texas Instruments, ZEUS de General Electric, etc., todos orientados al área industrial, así como los lenguajes en el ámbito universitario (AHPL, DDL, CDL, ISPS, etc.) [8]. Los primeros no estaban disponibles fuera de la empresa que los manejaba, mientras que los segundos carecían de soporte y mantenimiento adecuados que permitieran su utilización industrial. El desarrollo continuó y en la década de los ochenta surgieron lenguajes como VHDL, Verilog, ABEL 5.0, AHDL, etc., considerados lenguajes de descripción en hardware porque permitieron abordar un problema lógico a nivel funcional (descripción de un problema sólo conociendo las entradas y salidas), lo cual facilita la evaluación de soluciones alternativas antes de iniciar un diseño detallado.

Una de las principales características de estos lenguajes radica en su capacidad para describir en distintos niveles de abstracción (funcional, transferencia de registros RTL y lógico o nivel de compuertas) cierto diseño. Los niveles de abstracción se emplean para clasificar modelos HDL según el grado de detalle y precisión de sus descripciones [4].

Los niveles de abstracción descritos desde el punto de vista de simulación y síntesis del circuito pueden definirse como sigue:

- Algorítmico: se refiere a la relación funcional entre las entradas y salidas del circuito o sistema, sin hacer referencia a la realización final.
- Transferencia de registros (RT): Consiste en la partición del sistema en bloques funcionales sin considerar a detalle la realización final de cada bloque.
- Lógico o de compuertas: el circuito se expresa en términos de ecuaciones lógicas o de compuertas.

1.5.1 VHDL, lenguaje de descripción en hardware

En la actualidad, el lenguaje de descripción en hardware más utilizado a nivel industrial es VHDL[5] (Hardware Description Language), que apareció en

[5] Fuente de información: http://www.fgpa.com

la década de los ochenta como un lenguaje estándar, capaz de soportar el proceso de diseño de sistemas electrónicos complejos, con propiedades para reducir el tiempo de diseño y los recursos tecnológicos requeridos. El Departamento de Defensa de Estados Unidos creó el lenguaje VHDL como parte del programa "Very High Speed Integrated Circuits" (VHSIC), a partir del cual se detectó la necesidad de contar con un medio estándar de comunicación y la documentación para analizar la gran cantidad de datos asociados para el diseño de dispositivos de escala y complejidad deseados [9]; es decir, VHSIC debe entenderse como la rapidez en el diseño de circuitos integrados.

Después de varias versiones revisadas por el gobierno de los Estados Unidos, industrias y universidades, el IEEE (Instituto de Ingenieros Eléctricos y Electrónicos) publicó en diciembre de 1987 el estándar IEEEstd 1076-1987. Un año más tarde, surgió la necesidad de describir en VHDL todos los ASIC creados por el Departamento de Defensa, por lo que en 199.3 se adoptó el estándar adicional de VHDL IEEE1164.

Hoy en día VHDL se considera como un estándar para la descripción, modelado y síntesis de circuitos digitales y sistemas complejos. Este lenguaje presenta diversas características que lo hacen uno de los HDL más utilizados en la actualidad.

1.5.2 Ventajas del desarrollo de circuitos integrados con VHDL

A continuación se exponen algunas de las ventajas que representan los circuitos integrados con VHDL:

- Notación formal. Los circuitos integrados VHDL cuentan con una notación que permite su uso en cualquier diseño electrónico.
- Disponibilidad pública. VHDL es un estándar no sometido a patente o marca registrada alguna, por lo que cualquier empresa o institución puede utilizarla sin restricciones. Además, dado que el IEEE lo mantiene y documenta, existe la garantía de estabilidad y soporte.
- Independencia tecnológica de diseño. VHDL se diseñó para soportar diversas tecnologías de diseño (PLD, FPGA, ASIC, etc.) con distinta funcionalidad (circuitos combinacionales, secuenciales, síncronos y asincronos), a fin de satisfacer las distintas necesidades de diseño.
- Independencia de la tecnología y proceso de fabricación. VHDL se creó para que fuera independiente de la tecnología y del proceso de fabricación del circuito o del sistema electrónico. El lenguaje funciona de igual manera en circuitos diseñados con tecnología MOS, bipolares, BICMOS, etc., sin necesidad de incluir en el diseño información

concreta de la tecnología utilizada o de sus características (retardos, consumos, temperatura, etc.), aunque esto puede hacerse de manera opcional.

- Capacidad descriptiva en distintos niveles de abstracción. El proceso de diseño consta de varios niveles de detalle, desde la especificación hasta la implementación final (niveles de abstracción). VHDL ofrece la ventaja de poder diseñar en cualquiera de estos niveles y combinarlos, con lo cual se genera lo que se conoce como simulación multinivel.
- Uso como formato de intercambio de información. VHDL permite el intercambio de información a lo largo de todas las etapas del proceso de diseño, con lo cual favorece el trabajo en equipo.
- Independencia de los proveedores. Debido a que VHDL es un lenguaje estándar, permite que las descripciones o modelos generados en un sitio sean accesibles desde cualquier otro, sean cuales sean las herramientas de diseño utilizadas.
- Reutilización del código. El uso de VHDL como lenguaje estándar permite reutilizar los códigos en diversos diseños, sin importar que hayan sido generados para una tecnología (CMOS, bipolar, etc.) e implementación (FPGA, ASIC, etc.) en particular.
- Facilitación de la participación en proyectos internacionales. En la actualidad VHDL constituye el lenguaje estándar de referencia a nivel internacional. Impulsado en sus inicios por el Departamento de Defensa de Estados Unidos, cualquier programa lanzado por alguna de las dependencias oficiales de ese país vuelve obligatorio su uso para el modelado de los sistemas y la documentación del proceso de diseño [111. Este hecho ha motivado que diversas empresas y universidades adopten a VHDL como su lenguaje de diseño.

En Europa la situación es similar, ya que en nuestros días la mayoría de las grandes empresas del ramo lo ha definido como el lenguaje de referencia en todas las tareas de diseño, modelado, documentación y mantenimiento de los sistemas electrónicos. De hecho, el número de usuarios de VHDL en Europa es mayor que en Estados Unidos, debido en gran parte a que resulta el lenguaje más común en la mayoría de los consorcios.

1.5.3 Desventajas del desarrollo de circuitos integrados con VHDL

Como se puede observar, VHDL presenta grandes ventajas; sin embargo, es necesario mencionar también algunas desventajas que muchos diseñadores consideran importantes:

- En algunas ocasiones, el uso de una herramienta provista por alguna compañía en especial tiene características adicionales al lenguaje, con lo que se pierde tin poco la libertad de diseño. Como método alternativo, se pretende que entre diseñadores que utilizan distintas herramientas exista una compatibilidad en sus diseños, sin que esto requiera un esfuerzo importante en la traducción del código.
- Debido a que VHDL es un lenguaje diseñado por un comité, presenta una alta complejidad, ya que se debe dar gusto a las diversas opiniones de los miembros de éste, por lo que resulta un lenguaje difícil de aprender para un novato.

1.5.4 VHDL en la actualidad

La actividad que se ha generado en torno a VHDL es muy intensa. En muchos países como España se han creado grupos de trabajo alrededor de dicho lenguaje y se realizan periódicamente conferencias, reuniones, etc., donde se presentan trabajos tanto en Estados Unidos (en el VIUF, VHDL International User's Forum) como en Europa (VHDL Forum for CAD in Europe), así como en el congreso EuroVHDL celebrado desde 1992[10].

La participación europea en el esfuerzo de estandarizar el lenguaje se canaliza a través del proyecto ESPRIT, encabezado por SIEMENS-NIXDORF. En el proyecto participan prácticamente todas las grandes compañías europeas del sector electrónico, como ANACAD, ICL, PHILLIPS, TGI y THOMSON-CSF, además de diversas universidades y centros de investigación. Otras empresas dedicadas a la microelectrónica se han ido adaptando poco a poco al lenguaje. Incluso en Japón está teniendo una gran aceptación, no obstante que cuentan con un lenguaje estándar propio llamado UDL/I.

El proceso de estandarización del VHDL no se detuvo con la primera versión del lenguaje (VHDL'87), sino que ha continuado con la nueva versión (VHDL'93) y constantes actualizaciones, mejoras y metodologías de uso. Entre estas adiciones o actualizaciones se encuentra una muy importante: la ex' tensión analógica (1076.1), que permite la utilización de un lenguaje único en todas las tareas de especificación, simulación y síntesis de sistemas electrónicos digitales, analógicos o mixtos.

1.6 Compañías de soporte en hardware y software

Existen diversas compañías internacionales que fabrican o distribuyen dispositivos lógicos programables. Algunas ofrecen productos con características generales y otras introducen innovaciones a sus dispositivos. A continuación se mencionan algunas de las más importantes.

Altera Corporation

Altera es una de las compañías más importantes de producción de dispositivos lógicos programables y también es la que más familias ofrece, ya que tiene en el mercado ocho familias: APEX™20K, FLEX®10K, FLEX 8000, FLEX 6000, MAX® 9000, MAX7000, MAX5000, y Classic™. La capacidad de integración en cada familia varía desde 300 hasta 1 000 000 de compuertas utilizables por dispositivo, además de que todas tienen la capacidad de integrar sistemas complejos.

Las características generales más significativas de los dispositivos Altera son las siguientes:

- Frecuencia de operación del circuito superior a los 175 Mhz y retardos pin a pin de menos de 5 ns.
- La implementación de bloques de arreglos integrados (EAB), que se usan para realizar circuitos que incluyan funciones aritméticas como multiplicadores, ALU, y secuenciadores. También se aplican en micro-. procesadores, microcontroladores y funciones complejas con DSP (procesadores digitales de señales) [12].
- La programación en sistema (ISP), que permite programar los dispositivos montados en la tarjeta (Fig. 1.19).

En la figura 1.19a observamos la programación en sistema; es decir, no hay que retirar el circuito de la tarjeta para programarlo. En la figura 1.19b se muestra lo contrario: en este caso el tipo de programación es similar a la grabación cotidiana que realizamos, debido a que se debe colocar y quitar el dispositivo todas las veces que se quiera programar.

a) **b)**

Figura 1.19 a) Programación en sistema, **b)** Programación en montaje.

- Más de cuarenta tipos y tamaños de encapsulados, incluyendo el TQFP (thin quad fíat pack), el cual es un dispositivo delgado, de forma cuadrangular y plano, que permite ahorrar un espacio considerable en la tarjeta.
- Operación multivoltaje, entre los 5 y 3.3 volts, para máximo funcionamiento y 2.5 V en sistemas híbridos.

- Potentes herramientas de software como MAX + PLUS II, que soporta todas las familias de dispositivos de Altera, así como el software estándar compatible con VHDL.

Cypress semiconductor

La compañía Cypress Semiconductor ofrece una amplia variedad de dispositivos lógicos programables complejos (CPLD), que se encuentran en las familias Ultra37000™ y FLASH370i™. Cada una de estas familias ofrece la reprogramación en sistema (ISR), la cual permite reprogramar los dispositivos las veces que se quiera dentro de la tarjeta.

Todos los dispositivos de ambas familias trabajan con voltajes de operación de 5 o de 3.3 V y en su interior contienen desde 32 hasta 128 macroceldas.

En lo que respecta a software de soporte, Cypress ofrece su poderoso programa Warp, el cual se basa en VHDL. Este programa permite simular de manera gráfica el circuito programado, generando un archivo de mapa de fusibles (jedec) que puede ser programado directamente en cualquier PLD, CPLD o FPGA de Cypress o de otra compañía que sea compatible.

Clear logic

La compañía Clear Logic introdujo en noviembre de 1998 los dispositivos lógicos procesados por láser (LPDL), tecnología que provee reemplazos de los dispositivos de la Compañía Altera, pero a un costo y tamaño menores. La tecnología LPLD puede disponer de arriba de un millón de transistores para construir alrededor de 512 macroceldas. Sustituye al dispositivo MAX 7512A de Altera y reduce el tamaño más de 60% respecto al chip original. Las primeras familias introducidas por Clear Logic son CL7000 y CL7000E, las cuales tienden a crecer en un futuro.

Motorola

Motorola, empresa líder en comunicaciones y sistemas electrónicos, ofrece también dispositivos FPGA y FPAA (Field Programmable Array Analog: campos programables de arreglos analógicos). Los FPAA son los primeros campos programables para aplicaciones analógicas, utilizados en las áreas de transporte, redes, computación y telecomunicaciones.

Xilinx

Xilinx es una de las compañías líder en soluciones de lógica programable, incluyendo circuitos integrados avanzados, herramientas en software para diseño, funciones predefinidas y soporte de ingeniería. Xilinx fue la compañía que inventó los FPGA y en la actualidad sus dispositivos ocupan más de la mitad del mercado mundial de los dispositivos lógicos programables.

Los dispositivos de Xilinx reducen de manera significativa el tiempo requerido para desarrollar aplicaciones en las áreas de computación, telecomunicaciones, redes, control industrial, instrumentación, aplicaciones militares y para el consumo general.

Las familias de CPLD XC9500 y XC9500XL proveen una larga variedad de dispositivos programables con características que van desde los 5 a 3.3 volts de operación, 36 a 288 macroceldas, 34 a 192 terminales de entrada y salida, y programación en sistema.

Los dispositivos de las familias XC4000 y XC1700 de FPGA manejan voltajes de operación entre los 5 y 3.3 V, una capacidad de integración arriba de las 40 000 compuertas y programación en sistema.

En lo que se refiere a software, Xilinx desarrolló una importante herramienta llamada Foundation Series, que soporta diseños estándares basados en ABEL-HDL y en VHDL. Esta herramienta se ofrece en versión estudiantil y profesional.

De manera general, existe una amplia y variada gama de dispositivos lógicos programables disponibles en el mercado. La elección de uno u otro depende de los recursos con que cuenta el diseñador y los requerimientos del diseño. En la tabla 1.7 se muestran de forma simplificada algunas de las compañías que ofrecen soluciones de lógica programable, mientras que en la figura 1.20 se presentan sus productos.

Futuro de la lógica programable

Debido al auge actual de la lógica programable, no es difícil suponer que se pretende mejorar a futuro las herramientas existentes con el fin de extender su campo de aplicación a más áreas. Algunas compañías buscan mejorar la funcionalidad e integración de sus circuitos a fin de competir con el mercado de los ASIC. Esto mejoraría el costo por volumen, el ciclo de diseño y se disminuiría el voltaje de consumo.

Otra característica que se pretende mejorar es la reprogramación de los circuitos, debido a que su implementación requiere muchos recursos físicos y tecnológicos. Por esta razón se busca cambiar las metodologías de diseño para incluir sistemas reprogramables por completo.

Algunos desarrollos cuentan con memoria RAM o microprocesadores integrados en la tarjeta de programación. La tendencia de algunos fabricantes es integrar estos recursos en un circuito.

Compañía	Productos de hardware	Herramientas software
Altera	FPGA: Familias APEX 20K, FLEX 10K, FLEX 6000, MAX 9000, MAX 7000, MAX 5000 y CLASSIC	MAX + PLUS II: Soporta VHDL, Verilog y entrada esquemática.
Chip Express	LPGA (Laser Program Gate Array): CX3000, CX2000 y QYH500	QulCk Place&route: Diseños en base a vectores de prueba (CTV , ChipExpress Test Vector) y VHDL
Clear Logic	LPLD (Laser-processed Logic Device): CL7000, CL7000E, CL7000S	Desarrollos basados en herramientas de Altera.
Cypress Semiconductors	PLD: GAL22V10 CPLD: Ultra.37000, FLASH370Í	WARP: Soporta VHDL, Verilog y esquemáticos.
Motorola	FPAA (Field Programmable Analog Array): MPAA020	Easy Analog: herramienta de diseño interactiva exclusiva para diseño con FPAA.
Vantis	FPGA: Familias MACH4 y MACH5A	MACHXL: VHDL y Verilog
Quick Logic	PAsic (Asic Programable) y la familia QL de FPGA.	Quick Works: herramienta de soporte para VHDL, Verilog y captura esquemática.
Xilinx	CPLD: Familia XC9500 y XC9500XL FPGAs Familia XC400 y XC1700	Xilinx Foundation Series: soporta ABEL-HDL, VHDL y esquemáticos.

Tabla 1.7 Compañías de Soporte de lógica programable.

Figura 1.20 Dispositivos lógicos programables.

Ejercicios

1.1 ¿Qué significa monolítico?
1.2 ¿Cuál es el significado de las siglas ASIC?
1.3 ¿Cuáles son las categorías de tecnologías de fabricación de CI?
1.4 Describa en qué consiste el diseño Full Custom.
1.5 Mencione las características más relevantes del diseño Full Custom.
1.6 ¿Cuál es el significado de las siglas PLD?
1.7 ¿Qué tienen en común los dispositivos PROM, PLA, PAL, GAL y los CPLD y FPGA?
1.8 ¿Qué es OLMC?
1.9 ¿Cuál es el significado de las siglas CPLD y FPGA?
1.10 Describa cómo se encuentra estructurado un CPLD.
1.11 Describa la estructura de un FPGA en términos generales.
1.12 ¿Qué es un compilador lógico?
1.13 ¿Cuál es el significado de las siglas VHDL?
1.14 ¿Qué significado tienen las siglas VHSIC?
1.15 Describa tres ventajas de la programación en VHDL.
1.16 ¿Cuáles son las compañías más importantes en la fabricación de dispositivos lógicos programables?

Bibliografía

Maxinez David G., Alcalá Jessica: *Diseño de Sistemas Embebidos a través del Lenguaje de Descripción en Hardware VHDL.* XIX Congreso Internacional Académico de Ingeniería Electrónica. México, 1997.

Kloos C., Cerny E.: *Hardware Description Language and their aplications. Specification, modelling, verification and synthesis of microelectronic systems.* Chapman&Hall, 1997.

IEEE: *The IEEE standard VHDL Language Reference Manual.* IEEE-Std-1076-1987,1988.

Advanced Micro Devices: *Programmable Logic Handbook/Data book.* Advanced Micro Devices, 1986.

Zainalabedin Navabi: *Analysis and Modeling of Digital Systems.* McGraw-Hill, 1988.

Altera Corporation: *User Configurable Logic Data Book.* Altera Corp., 1988.

Altera Corporation: *The Maximalist Handbook.* Altera Corp., 1990.

Ismail M., Fiez T.: *Analog VLSI.* McGraw-Hill, 1994.

Hayes John E: *Computer Architecture and Organization.* McGraw-Hill, 1979.

Wakerly J. F.: *Digital Desing Principles and practices.* Prentice Hall, 1990.

Skahill Kevin.: *VHDL for programmable logic.* Addison Wesley, 1996.

Cypress Corporation: www.cypress.com

Xilinx Corporation: www.xilinx.com

Organización Mundial de VHDL: www.vhdl.org

Campos de lógica programable: www.fpga.com

Referencias

[1] Maxinez G. David: *Amplificación de Señales.* ITESM-CEM, 1993.

[21 Hon R. W. y Sequin C.H.: A *guide to LSI implentation.* Xerox Parc, 1980.

[31 Mead C. y Conway L.: *Introduction to VLSI Systems.* Addison Wesley, VLSI series 1980.

[4] Teres LI., Torroja Y., Olcoz S., Villar E.: *VHDL Lenguaje Estándar de Diseño Electrónico.* McGraw-Hill, 1998.

[5] Floyd T. L.: *Fundamentos de Sistemas Digitales.* Prentice Hall, 1998.

[6, 7] Van den Bout Dave: *The practical Xilinx Designer Lab Book.* Prentice Hall, 1998.

[8] Instituto de Ingeniería Eléctrica y Electrónica, IEEE. *Revista Computer.* IEEE, 1977.

[9] Delgado C., Lecha E., Moré M., Terés Ll., Sánchez L.: *Introducción a los lenguajes VHDL, Verilogy* UDL/Í. Novática No. 112, España, 1993.

[9] Delgado C., Lecha E., Moré M., Terés LL, Sánchez L.: *Introducción a los lenguajes VHDL, Verilog y UDL/I.* Novática No. 112, España, 1993.
[10] Ecker W.: The Design Cube. Euro VHDL Forum, 1995.
[11] Novatica (varios autores): *Monografía sobre los lenguc !es de diseño de hardware.* Revista Novatica, núms. 112-113, nov-94 a leb-95.
[12] Altera Corporation: www.altera.com

Capítulo 2

VHDL: su organización y arquitectura

Introducción

Tal como lo indican sus siglas, VHDL (Hardware Description Language) es un lenguaje orientado a la descripción o modelado de sistemas digitales; es decir, se trata de un lenguaje mediante el cual se puede describir, analizar y evaluar el comportamiento de un sistema electrónico digital.

VHDL es un lenguaje poderoso que permite la integración de sistemas digitales sencillos, elaborados o ambos en un dispositivo lógico programable, sea de baja capacidad de integración como un GAL, o de mayor capacidad como los CPLD y FPGA.

2.1 Unidades básicas de diseño

La estructura general de un programa en VHDL está formada por módulos o unidades de diseño, cada uno de ellos compuesto por un conjunto de declaraciones e instrucciones que definen, describen, estructuran, analizan y evalúan el comportamiento de un sistema digital.

Existen cinco tipos de unidades de diseño en VHDL: declaración de entidad (entity declaration), arquitectura (architecture), configuración (configuration), declaración del paquete (package declaration) y cuerpo del paquete (package body). En el desarrollo de programas en VHDL pueden utilizarse o no tres de los cinco módulos, pero dos de ellos (entidad y arquitectura) son indispensables en la estructuración de un programa.

Las declaraciones de entidad, paquete y configuración se consideran unidades de diseño primarias, mientras que la arquitectura y el cuerpo del paquete son unidades de diseño secundarias porque dependen de una entidad primaria que se debe analizar antes que ellas.

2.2 Entidad

Una entidad (*entity*) es el bloque elemental de diseño en VHDL, Las entidades son todos los elementos electrónicos (sumadores, contadores, compuertas, flip-flops, memorias, multiplexores, etc.) que forman de manera individual o en conjunto un sistema digital. La entidad puede representarse de muy diversas maneras; por ejemplo, la figura 2.1a) muestra la arquitectura de un sumador completo a nivel de compuertas; ahora bien, esta entidad se puede representar a nivel de sistema indicando tan sólo las entradas (Cin, A y B) y salidas (SUMA y Cout) del circuito: figura 2.1b). De igual forma, la integración de varios subsistemas (medio sumador) puede representarse mediante una entidad [Fig. 2.1c)]. Los subsistemas pueden conectarse internamente entre sí; pero la entidad sigue identificando con claridad sus entradas y salidas generales.

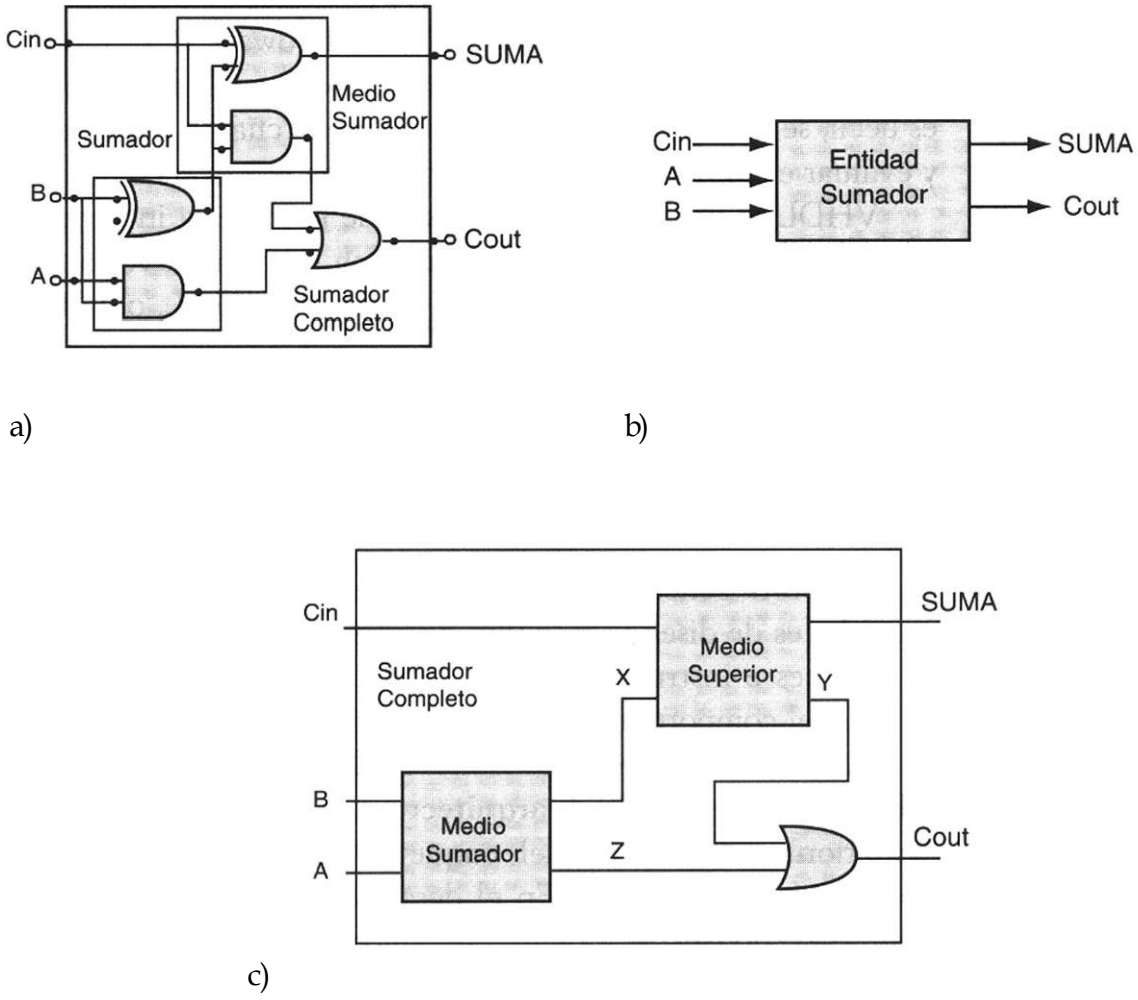


Figura 2.1 a) Descripción a nivel de compuertas, **b)** Símbolo funcional de la entidad; c) Diagrama a bloques representativo de la entidad.

2.2.1 Puertos de entrada'salida

Cada una de las señales de entrada y salida en una entidad son referidas como puerto, el cual es similar a una terminal (pin) de un símbolo esquemático. Todos los puertos que son declarados deben tener un nombre, un modo y un tipo de dato. El nombre se utiliza como una forma de llamar al puerto; el modo permite definir la dirección que tomará la información y el tipo define qué clase de información se transmitirá por el puerto. Por ejemplo, respecto a los puertos de la entidad que representan a un comparador de igualdad (Fig. 2.2), las variables a y b denotan los puertos de entrada y la variable c se refiere al puerto de salida.

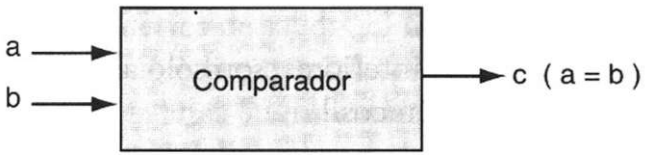


Figura 2.2 Comparador de igualdad.

2.2.2 Modos

Como ya se mencionó, un modo permite definir la dirección en la cual el dato es transferido a través de un puerto. Un modo puede tener uno de cuatro valores: in (entrada), out (salida), inout (entrada/salida) y buffer (Fig. 2.3).

- Modo in. Se refiere a las señales de entrada a la entidad. Este sólo es unidireccional y nada más permite el flujo de datos hacia dentro de la entidad.
- Modo out. Indica las señales de salida de la entidad.
- Modo inout. Permite declarar a un puerto de forma bidireccional —es decir, de entrada/salida—; además permite la retroalimentación de señales dentro o fuera de la entidad.
- Modo buffer. Permite hacer retroalimentaciones internas dentro de la entidad, pero a diferencia del modo inout, el puerto declarado se comporta como una terminal de salida.

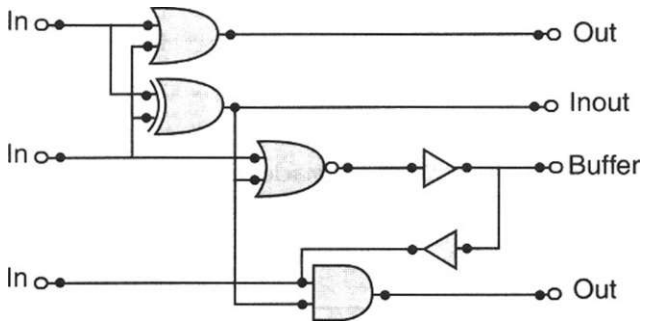


Figura 2.3 Modos y el curso de sus señales.

2.2.3 Tipos de datos

Los tipos son los valores (datos) que el diseñador establece para los puertos de entrada y salida dentro de una entidad; se asignan de acuerdo con las características de un diseño en particular. Algunos de los tipos más utilizados en VHDL son:

- **Bit,** el cual tiene valores de 0 y 1 lógico.
- **Boolean** (booleano) que define valores de verdadero o falso en una expresión
- **Bit_vector** (vectores de bits) que representa un conjunto de bits para cada variable de entrada o salida.
- **Integer** (entero) que representa un número entero.

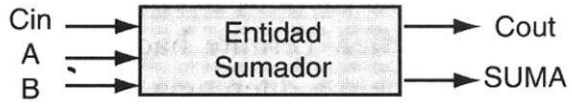
Los anteriores son sólo algunos de los tipos que maneja VHDL, pero no son los únicos.[1]

2.3 Declaración de entidades

Como se mencionó en la sección 2.1 (Unidades básicas de diseño), los módulos elementales en el desarrollo de un programa dentro del lenguaje de descripción en hardware (VHDL) son la entidad y la arquitectura.

La declaración de una entidad consiste en la descripción de las entradas y salidas de un circuito de diseño identificado como entity (entidad); es decir, la declaración señala las terminales o pines de entrada y salida con que cuenta la entidad de diseño.

Por ejemplo, la forma de declarar la entidad correspondiente al circuito sumador de la figura 2.1b) se muestra a continuación:




```
1   —Declaración de la entidad de un circuito sumador
2   entity sumador is
3   port    (A, B, Cin:    in bit;
4           SUMA, Cout: out bit);
5   end sumador;
```

Listado 2.1 Declaración de la entidad sumador de la figura 2.1b).

[1] En el apéndice A se listan los tipos de datos existentes en VHDL.

Los números de las líneas (1, 2, 3, 4, 5) no son parte del código; se usan como referencia para explicar alguna sección en particular. Las palabras en negritas están reservadas para el lenguaje de programación VHDL; esto es, tienen un significado especial para el programa; el diseñador asigna los otros términos.

Ahora comencemos a analizar el código línea por línea. Observemos que la línea 1 inicia con dos guiones (--), los cuales indican que el texto que está a la derecha es un comentario cuyo objetivo es documentar el programa, ya que el compilador ignora todos los comentarios. En la línea 2 se inicia la declaración de la entidad con la palabra reservada entity, seguida del identi* ficador o nombre de la entidad (sumador) y la palabra reservada is. Los puertos de entrada y salida (port) se declaran en las líneas 3 y 4, respectivamente —en este caso los puertos de entrada son A, B y Cin—, mientras que SUMA y Cout representan los puertos de salida. El tipo de dato que cada puerto maneja es del tipo bit, lo cual indica que sólo pueden manejarse valores de '0' y '1' lógicos. Por último, en la línea 5 termina la declaración de entidad con la palabra reservada end, seguida del nombre de la entidad (sumador).

Debemos notar que como cualquier lenguaje de programación, VHDL sigue una sintaxis y una semántica dentro del código, mismas que hay que respetar. En esta entidad conviene hacer notar el uso de punto y coma (;) al finalizar una declaración y de dos puntos (:) al asignar nombres a las entradas y salidas.

Ejemplo 2.1 Declare la entidad del circuito lógico de la figura C2.1.

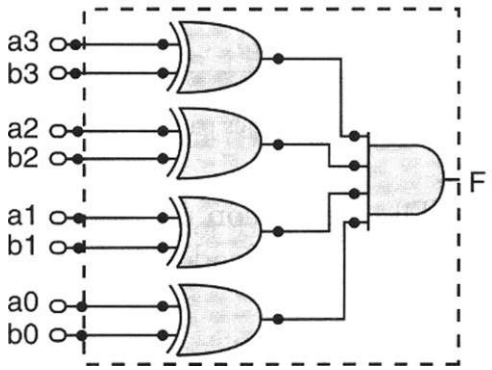


Figura E2.1

Solución

Como puede observarse, las entradas y salidas del circuito se encuentran delimitadas por la línea punteada. En este caso, a3, b3, a2, b2,... aO, bO son las entradas y F es la salida.

La declaración de la entidad sería de la siguiente forma:

```
1   --  Declaración de la entidad
2   Entity circuito is
3       port( a3,b3,a2,b2,a1,b1,a0, b0:  in bit;
4                                    F:  out bit);
5       end circuito;
```

Identificadores

Los identificadores son simplemente los nombres o etiquetas que se usan para referir variables, constantes, señales, procesos, etc. Pueden ser números, letras del alfabeto y guiones bajos (_) que separen caracteres y no tienen una restricción en cuanto a su longitud. Todos los identificadores deben seguir ciertas especificaciones o reglas para que se puedan compilar sin errores, mismas que aparecen en la tabla 2.1.

Regla	Incorrecto	Correcto
El primer caracter siempre es una letra mayúscula o minúscula.	4suma	Suma4 SUMA4
El segundo caracter no puede ser un guión bajo	S_4bits	S4_bits
Dos guiones juntos no son permitidos	Resta__4	Resta_4_
Un identificador no puede utilizar símbolos	Clear#8	Clear_8

Tabla 2.1 Especificaciones para la escritura de identificadores.

VHDL cuenta con una lista de palabras reservadas que no pueden funcionar como identificadores (vea el Apéndice B).

2.4 Diseño de entidades mediante vectores

La entidad sumador realizada en el circuito del listado 2.1, usa bits individuales, los cuales sólo pueden representar dos valores lógicos (0 o 1). De manera general, en la práctica se utilizan conjuntos (palabras) de varios bits; en VHDL las palabras binarias se conocen como vectores de bits, los cuales se consideran un grupo y no como bits individuales. Como ejemplo considérense los vectores de 4 bits que se muestran a continuación:

```
vector_A      =  [A3, A2, A1, A0]
vector_B      =  [B3, B2, B1, B0]
vector_SUMA   =  [S3, S2, S1, S0]
```

En la figura 2.4 se observa la entidad del sumador analizado antes, sólo que ahora las entradas A, B y la salida SUMA incorporan vectores de 4 bits en sus puertos. Obsérvese cómo la entrada Cin y la salida Cout son de un bit.

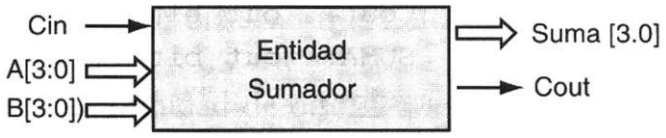


Figura 2.4 Entidad representada por vectores.

La manera de describir en VHDL una configuración que utilice vectores consiste en la utilización de la sentencia bit_vector, mediante la cual se especifican los componentes de cada uno de los vectores utilizados. La parte del código que se usa para declarar un vector dentro de los puertos es el siguiente:

```
port (vector_A, vector_B:  in  bit_vector   (3 downto 0);
            vector_SUMA:  out  bit_vector (3 downto 0));
```

Esta declaración define los vectores (A, B y SUMA) con cuatro componentes distribuidos en orden descendente por medio del comando:

3 downto 0 (3 hacia 0)

los cuales se agruparían de la siguiente manera.

```
vector_A(3) = A3      vector_B(3) = B3      vector_SUMA(3) = S3
vector_A(2) = A2      vector_B(2) = B2      vector_SUMA(2) = S2
vector_A(l) = Al      vector_B(l) = B1      vector_SUMA(l) = SI
vector A(0) = A0      vector_B(0) = B0      vector SUMA(O) = SO
```

Una vez que se ha establecido el orden en que aparecerán los bits enunciados en cada vector, no se puede modificar, a menos que se utilice el comando to:

Oto 3 (0 hasta 3)

que indica el orden de aparición en sentido ascendente.

Ejemplo 2.2 Describa en VHDL la entidad del circuito sumador representado en la figura 2.4. Observe cómo la entrada Cin (Carry in) y la salida Cout (Carry out) se expresan de forma individual.

Solución

```
entity sumador is
port   (A,B:    in bit_vector (3 downto 0);
         Cin:    in bit;
        Cout:   out bit;
        SUMA:   out bit_vector(3 downto 0));
end sumador;
```

Ejemplo 2.3 Declare la entidad del circuito lógico mostrado en la figura del ejemplo 2.1, mediante vectores.

Solución

```
1 -Declaración de entidades mediante vectores
2 entity detector is
3 port  (a,b:  in bit_vector(3 downto 0);
4           F:  out bit));
5 end detector;
```

2.4.1 Declaración de entidades mediante librerías y paquetes

Una parte importante en la programación con VHDL radica en el uso de li' brerías y paquetes que permiten declarar y almacenar estructuras lógicas, seccionadas o completas que facilitan el diseño.

Una librería o biblioteca es un lugar al que se tiene acceso para utilizar las unidades de diseño predeterminadas por el fabricante de la herramienta (paquete) y su función es agilizar el diseño. En VHDL se encuentran definidas dos librerías llamadas ieee y work (Fig. 2.5). Como puede observarse, en la librería ieee se encuentra el paquete std_logic_l 164, mientras que en la librería work se hallan numeric_std, std_arith y gatespkg.

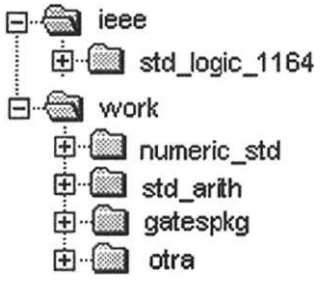


Figura 2.5 Contenido de las librerías ieee y work.

En una librería también se permite almacenar el resultado de la compilación de un diseño, con el fin de utilizar en uno o varios programas. La librería work es el lugar establecido donde se almacenan los programas que el usuario va generando. Esta librería se encuentra siempre presente en la compilación de un diseño y los diseños se guardan en ella mientras no se especifique otra. La carpeta otra mostrada en la figura 2.5 representa esta situación.

Un paquete es una unidad de diseño que permite desarrollar un programa en VHDL de una manera ágil, debido a que contiene algoritmos preestablecidos (sumadores, restadores, contadores, etc.) que ya tienen optimizado su comportamiento. Por esta razón, el diseñador no necesita caracterizar paso a paso una nueva unidad de diseño si ya se encuentra almacenada en algún paquete —en cuyo caso basta con llamarla y especificarla en el programa—. Por lo tanto, un paquete no es más que una unidad de diseño formada por declaraciones, programas, componentes y subprogramas, que incluyen los diversos tipos de datos (bit, booleano, std_logic), empleados en la programación en VHDL y que suelen formar parte de las herramientas en software.

Por último, cuando en el diseño se utiliza algún paquete es necesario llamar a la librería que lo contiene. Para esto se utiliza la siguiente declaración:

library ieee;

Lo anterior permite el uso de todos los componentes incluidos en la librería **ieee.** En el caso de la librería de trabajo **(work),** su uso no requiere la declaración **library,** dado que la carpeta work siempre está presente al desarrollar un diseño.

2.4.2 Paquetes

- El paquete **std_logic_1164** (estándar lógico_1164) que se encuentra en la librería ieee contiene todos los tipos de datos que suelen emplearse en VHDL (std_logic_vector, std_logic, entre otros).

El acceso a la información contenida en un paquete es por medio de la sentencia **use,** seguida del nombre de la librería y del paquete, respectivamente:

use nombre_librería.nombre_paquete**.all;**

por ejemplo:

use ieee.std_logic_1164**.all;**

En este caso ieee es la librería, std_logic_l 164 es el paquete y la palabra reservada all indica que se pueden usar todos los componentes almacenados en el paquete.

- El paquete numeric_std define funciones para realizar operaciones entre diferentes tipos de datos (sobrecargado); además, los tipos pueden representarse con signo o sin éste (vea el Apéndice A).
- El paquete numeric_bit define tipos de datos binarios con signo o sin éste.
- El paquete std_arith define funciones y operadores aritméticos, como igual (=), mayor que (>), menor que (<), entre otros (vea el Apéndice A).

En lo sucesivo, se usarán a menudo las librerías y paquetes de los programas desarrollados en el texto.

Ejemplo 2.4 En la figura E2.4 se muestra el bloque representativo de un circuito multiplicador de 2 bits. La multiplicación de (XI, XO) y (Y1,Y0) producen la salida Z3, Z2, Z1, ZO.

Declare la entidad del circuito utilizando la librería ieee y el paquete std_logic_1164.all.

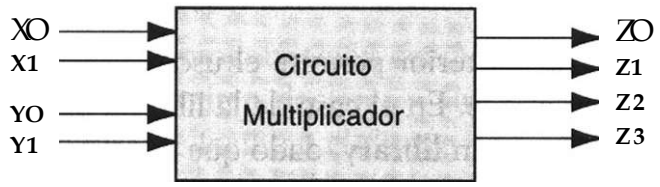


Figura E2.4

Solución

```
1 library ieee;
2 use  ieee.std_logic_1164.all;
3 entity multiplica is
4 port  (X0,X1,YO,Yl:  in std_logic;
6        Z3,Z2,Z1,Z0:  out std_logici
7 end multiplica;
```

2.5 Arquitecturas

Una arquitectura (*architecture*) se define como la estructura que describe el funcionamiento de una entidad, de tal forma que permita el desarrollo de los procedimientos que se llevarán a cabo con el fin de que la entidad cumpla las condiciones de funcionamiento deseadas.

La gran ventaja que presenta VHDL para definir una arquitectura radica en la manera en que pueden describirse los diseños; es decir, mediante el algoritmo de programación empleado se puede describir desde el nivel de compuertas hasta sistemas complejos.

De manera general, los estilos de programación utilizados en el diseño de arquitecturas se clasifican como:

- Estilo funcional
- Estilo por flujo de datos
- Estilo estructural

El nombre asignado a estos estilos no es importante, ya que es tarea del diseñador escribir el comportamiento de un circuito utilizando uno u otro estilo que a su juicio le sea el más acertado.

2.5.1 Descripción funcional

En la figura 2.6 se describe funcionalmente el circuito comparador. Se trata de una descripción funcional porque expone la forma en que trabaja el sistema; es decir, las descripciones consideran la relación que hay entre las entradas y las salidas del circuito, sin importar cómo esté organizado en su interior. Para este caso:

si a = b entonces c = 1
si a ≠ b entonces c = 0

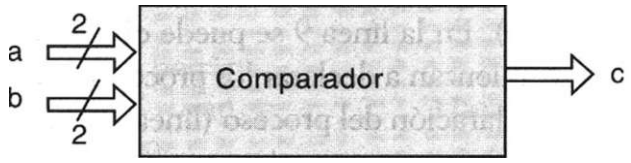


Figura 2.6 Descripción funcional de un comparador de igualdad de dos bits.

El código que representa el circuito de la figura 2.6 se muestra en el listado 2.2:

```
1     -- Ejemplo de una descripción funcional
2     library ieee;
3     use ieee.std_logic_1164.all;
4     entity comp is
5     port (a,b: in bit_vector( 1 downto 0);
6              c: out bit);
7     end comp;
8     architecture funcional of comp is
9     begin
10    compara: process (a,b)
11    begin
12         if a = b then
13              c <='1';
14         else
15              c<='0';
16    end if;
17    end process compara;
18    end funcional;
```

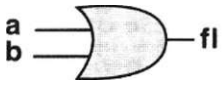
Listado 2.2 Arquitectura funcional de un comparador de igualdad de 2 bits.

Nótese cómo la declaración de la entidad (entity) se describe en las líneas de la 1 a la 7; el código ocupa de la línea 8 a la 18, donde se desarrolla el algoritmo **(architecture)** que describe el funcionamiento del comparador. Para iniciar la declaración de la arquitectura (línea 8), es necesario definir un nombre arbitrario con que se pueda identificar —en nuestro caso el nombre asignado fue funcional— además de incluir la entidad con que se relaciona (comp). En la línea 9 se puede observar el inicio **(begin)** de la sección donde se comienzan a declarar los procesos que rigen el comportamiento del sistema. La declaración del proceso (línea 10) se utiliza para la definición de algoritmos y comienza con una etiqueta opcional (en este caso compara), seguida de dos puntos (:), la palabra reservada **process** y une la lista sensitiva (a y b), que hace referencia a las señales que determinan el funcionamiento del proceso.

Al seguir el análisis, puede notarse que de la línea 12 a la 17 el proceso se ejecuta mediante declaraciones secuenciales del tipo **if-then'else** (si-entonces-si no). Esto se interpreta como sigue (línea 12): si el valor de la señal *a* es igual al valor de la señal *b,* **entonces** **'1'** se asigna a c, **si no** (else)se asigna un '0' (el símbolo < = se lee como "se asigna a"). Una vez que se ha definido el proceso, se termina con la palabra reservada **end process** y de manera opcional el nombre del proceso (compara); de forma similar se añade la etiqueta (funcional) al terminar la arquitectura en la línea 18.

Como se puede observar, la descripción funcional se basa principalmente en el uso de procesos y de declaraciones secuenciales, las cuales permiten modelar la función con rapidez.

Ejemplo 2.5 Describa mediante declaraciones del tipo if-then-else el funcionamiento de la compuerta OR mostrada en la figura E2.5 con base en la tabla de verdad.



a	b	f[
0	0	0
0	1	1
1	o	1
1	1	1

Figura E2.5

Solución

Como puede observarse, la declaración de la librería y el paquete se introducen en las líneas 2 y 3, respectivamente. La declaración de la entidad se define entre las líneas 4 a 7 inclusive. Por último, la arquitectura se describe en las líneas 8 a 17.

```
1  -- Declaración funcional
2  library ieee;
3  use ieee.std_logic_1164.all;
4  entity com_or is
5  port( a,b: in std_logic;
6          fl: out std_logic);
7  end com_or;
8  architecture funcional of com_or is
9  begin
10 process (a,b) begin
11     if (a = '0' and b = '0') then
12         fl <= '0';
13     else
14         fl <= '1';
15 end if;
16 end process;
17 end funcional;
```

2.5.2 Descripción por flujo de datos

La descripción por flujo datos indica la forma en que los datos se pueden transferir de una señal a otra sin necesidad de declaraciones secuenciales

(if-then-else). Este tipo de descripciones permite definir el flujo que tomarán los datos entre módulos encargados de realizar operaciones. En este tipo de descripción se pueden utilizar dos formatos: mediante instrucciones when-else (cuando-si no) o por medio de ecuaciones booleanas.

a) Descripción por flujo de datos mediante when-else

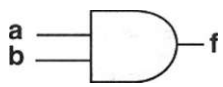
A continuación se muestra el código del comparador de igualdad de dos bits descrito antes (Fig. 2.6) Nótese que la diferencia entre los listados 2.2 y 2.3 radica en la eliminación del proceso y en la descripción sin declaraciones secuenciales (if-then-else).

```
1  --Ejemplo de declaración de la entidad de un comparador
2  library ieee;
3  use ieee.std_logic_1164.all ;
4  entity comp is
5  port  (a,b:   in bit_vector  (1 downto 0);
6           c:   out bit);
   end comp;
7
8  architecture f_datos of comp is
9  begin
10 c <= '1' when  (a = b)   else '0';  (asigna a C el valor
11 de 1 cuando a=b si no vale 0).
12 end  f_datos;
```

Listado 2.3 Arquitectura por flujo de datos.

En VHDL se manejan dos tipos de declaraciones: *secuenciales* y *concurrentes.* Una declaración secuencial de la forma if-then-else se halla en el listado 2.2 dentro del proceso, donde su ejecución debe seguir un orden para evitar la pérdida de la lógica descrita. En cambio, en una declaración concurrente esto no es necesario, ya que no importa el orden en que se ejecutan. Tal es el caso del listado 2.3.

Ejemplo 2.6 Con base en la tabla de verdad y mediante la declaración when-else, describa el funcionamiento de la siguiente compuerta AND.



a	b	fl
0	0	0
0	1	o
1	o	o
1	1	1

Figura E2.6

Solución

```
1 --Algoritmo utilizando flujo de datos
2 library ieee;
3 use ieee.std_logic_1164.all;
4 entity com_and is
5 port( a,b: in std_logic;
6           f: out std_logic);
7 end com_and;
8 architecture compuerta of com_and is
9 begin
10     f <= '1' when (a - '1' and b = '1' ) else
11          '0';
12 end compuerta;
```

b) Descripción por flujo de datos mediante ecuaciones booleanas

Otra forma de describir el circuito comparador de dos bits es mediante la obtención de sus ecuaciones booleanas figura 2.7. En el listado 2.4 se observa este desarrollo.

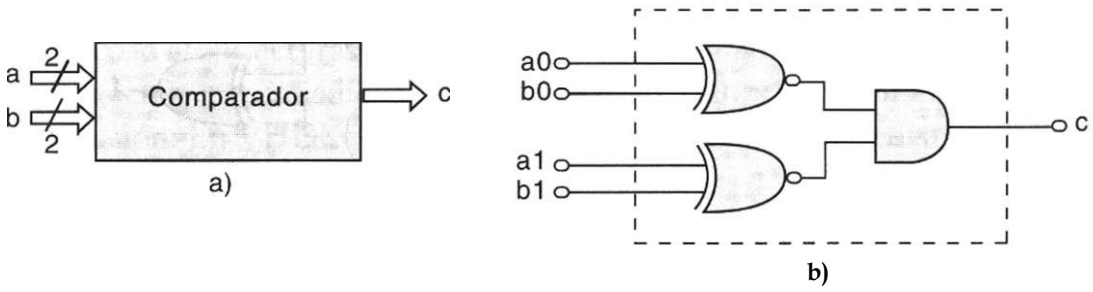


Figura 2.7 a) Entidad del comparador de dos bits, **b)** Comparador de dos bits realizado con compuertas.

El interior del circuito comparador de la figura 2.7a) puede representarse por medio de compuertas básicas [Fig. 2.7b)] y este circuito puede describirse mediante la obtención de sus ecuaciones booleanas.

```
1 -- Ejemplo de declaración de la entidad de un comparador
2  library ieee;
3 use  ieee.std_logic_1164.all;
4 entity comp is
5 port     ( a,b:    in bit_vector  (1 downto 0)  ;
6              c:    out bit)  ;
7 end comp;
8 architecture booleana of comp is
9 begin
10 c <=  (a(1)   xnor b(l)
11      and   a(0)  xnor b(0)) ;
12 end booleana;
```

Listado 2.4 Arquitectura de forma de flujo de datos construido por medio de ecuaciones booleanas.

La forma de flujo de datos en cualquiera de sus representaciones describe el camino que los datos siguen al ser transferidos de las operaciones efectuadas entre las entradas a y b a la señal de salida c.

Ejemplo 2,7 Describa mediante ecuaciones booleanas el circuito mostrado a continuación.

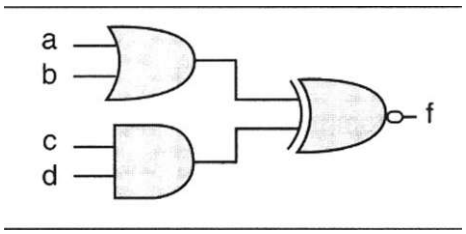


Figura E2.7

Solución

```
--Declaración mediante ecuaciones boolenas
library ieee;
use  ieee.std_logic_1164.all;
entity ejemplo is
port  ( a,b,c,d: in std_logic;
              f: out std_logic);
end ejemplo;
architecture compuertas of ejemplo is
begin
      f <=  ((a or b)  xnor  (c and b) ) ;
end  compuertas;
```

2.5.3 Descripción estructural

Como su nombre indica, una descripción estructural basa su comportamiento en modelos lógicos establecidos (compuertas, sumadores, contadores, etc.). Según veremos más adelante, el usuario puede diseñar estas estructuras y guardarlas para su uso posterior o tomarlas de los paquetes contenidos en las librerías de diseño del software que se esté utilizando.

En la figura 2.8 se encuentra un esquema del circuito comparador de igualdad de 2 bits, el cual está formado por compuertas ñor-exclusivas y una compuerta AND.

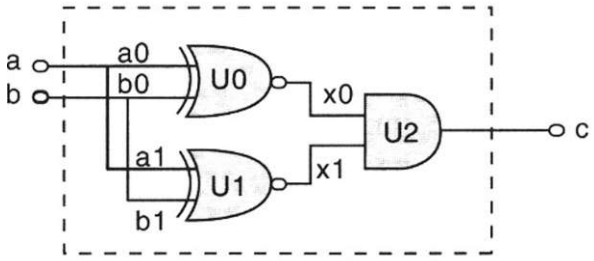


Figura 2.8 Representación esquemática de un comparador de 2 bits.

En nuestro caso, cada compuerta (modelo lógico) se encuentra dentro del paquete gatespkg,[6] del cual se toman para estructurar el diseño. A su vez, este tipo de arquitecturas estándares se conoce como componentes, que al interconectarse por medio de señales internas (xO, x1) permiten proponer una solución. En VHDL esta conectividad se conoce como netlist[7] o listado de componentes.

Para iniciar la programación de una entidad de manera estructural, es necesario la descomposición lógica del diseño en pequeños submódulos (jerarquizar), los cuales permiten analizar de manera práctica el circuito, ya que la función de entrada/salida es conocida. En nuestro ejemplo se conoce la función de salida de las dos compuertas xnor, por lo que al unirlas a la compuerta and, la salida c es el resultado de la operación and efectuada en el interior a través de las señales xO y xl (Fig. 2.8).

Es importante resaltar que una jerarquía en VHDL se refiere al procedimiento de dividir en bloques y no a que un bloque tenga mayor jerarquía (peso) que otro. Esta forma de dividir el problema hace de la descripción estructural una forma sencilla de programar. En el contexto del diseño lógico esto es observable cuando se analiza por separado alguna sección de un sistema integral.

[6] El paquete *compuerta* fue programado para este ejemplo. En el capítulo 8 se verá a detalle su desarrollo.
[7] Un netlist se refiere a la forma en como se encuentran conectados los componentes dentro de una estructura y las señales que propicia esta interconexión.

En el listado 2.5 se muestra el código del programa que representa al esquema de la figura 2.8.

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity comp is port (
4         a,b: in bit_vector (0 to 1);
5           c: out bit);
6 end comp;
7 use work.compuerta.all;
8 architecture estructural of comp is
9 signal x: bit_vector (0 to 1);
10 begin
11   U0: xnor2     port map (a(0), b(0),x(0)
12   Ul: xnor2     port map (a(l), b(1),x(1)
13   U2: and2      port map (x(0),x(l), c);

14 end estructural;
```

Listado 2.5 Descripción estructural de un comparador de igualdad de 2 bits.

En el código se puede ver que en la entidad nada más se describen las entradas y salidas del circuito (a, b y c), según se ha venido haciendo (líneas 3 a la 6). Los componentes xnor y and no se declaran debido a que se encuentran en el paquete de compuertas (gatespkg), el cual a su vez está dentro de la librería de trabajo (work), línea 7.

En la línea 8 se inicia la declaración de la arquitectura *estructural.* El algoritmo propuesto (líneas 11 a 13) describe la estructura de la siguiente forma: cada compuerta se maneja como un bloque lógico independiente (componente) del diseño original, al cual se le asigna una variable temporal (U0, U1 y U2); la salida de cada uno de estos bloques se maneja como una señal línea 9, signal x (xO y x1), las cuales se declaran dentro de la arquitectura y no en la entidad, debido a que no representan a una terminal (pin) y sólo se utilizan para conectar bloques de manera interna a la entidad.

Por último, podemos observar que la compuerta and recibe las dos señales provenientes de x (xO y x1), ejecuta la operación y asigna el resultado a la salida c del circuito.

Ejemplo 2.8 Realice el programa correspondiente en VHDL para el circuito mostrado en la figura E2.8. Utilice descripción estructural.

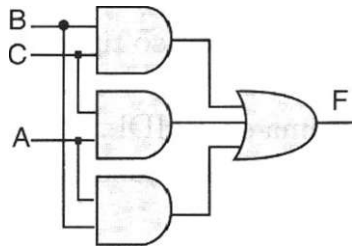


Figura E2.8

Solución

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity comp is
4 port( A,B,C : in std_logic;
5            F: out std_logic);
6 end comp;
7 use work.compuerta.all;
8 architecture estructura of comp is
9 signal x: bit_vector (0 to 2);
10 begin
11 U0: and2 port map (B, C, x(0));
12 U1: and2 port map (C, A, x(1));
13 U2: and2 port map (A, B, x(2));
14 U3: or3 port map (x(0), x(1), x(2), F);
15 end estructura;
```

Comparación entre los estilos de diseño

El estilo de diseño utilizado en la programación del circuito depende del diseñador y de la complejidad del proyecto. Por ejemplo, un diseño puede describirse por medio de ecuaciones booleanas, pero si es muy extenso quizá sea más apropiado emplear estructuras jerárquicas para dividirlo; ahora bien, si se requiere diseñar un sistema cuyo funcionamiento dependa sólo de sus entradas y salidas, es conveniente utilizar la descripción funcional, la cual presenta la ventaja de requerir menos instrucciones y el diseñador no necesita un conocimiento previo de cada componente del circuito.

Ejercicios

Unidades básicas de diseño

2.1 Describa los cinco tipos de unidades de diseño en VHDL.
2.2 Determine cuáles son las unidades de diseño necesarias para realizar un programa en VHDL.
2.3 Mencione las unidades de diseño primarias y secundarias.

Declaración de entidades

2.4 Describa el significado de una entidad y cuál es su palabra reservada.
2.5 En la siguiente declaración de entidad indique:

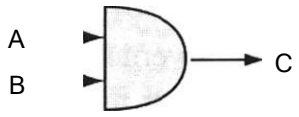
```
library ieee;
use ieee.std_logic_l 164.all;
entity selección is port (
x: in std_logic_vector(0 to 3);
f: out std_logic);
end selección;
```

a) El nombre de la entidad
b) Los puertos de entrada
c) Los puertos de salida
d) El tipo de dato

2.6 Señale cuáles de los siguientes identificadores son correctos o incorrectos, colocando en las líneas de respuesta la letra 'C' o T, respectivamente.

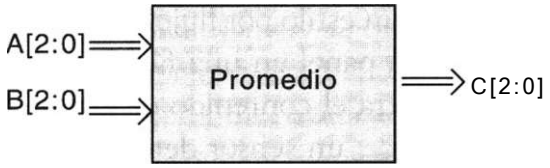
Ilógico_____Desp_laza
con_trol_____N_ivel
Página_____architecture
Registro_____S_uma#
2Suma Res ta

2.7 Declare la entidad para la compuerta AND del ejercicio 2.7:

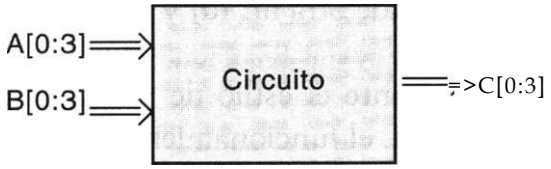


2.8 Declare la entidad para el siguiente circuito.

2.9 Declare la entidad para el circuito que se muestra en la figura. Utilice vectores.



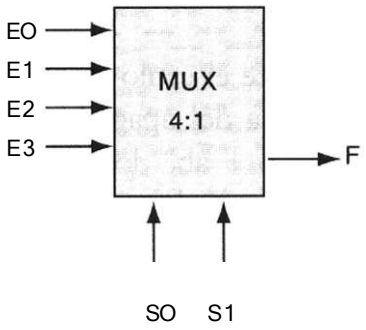
2.10 Declare la entidad para el siguiente circuito que utiliza vectores.



2.11 Describa qué es una librería en VHDL.

2.12 Indique el significado de la siguiente expresión:
use ieee.std_logic_1164.all;

2.13 Declare la entidad del circuito multiplexor de 4:1 mostrado en la figura del ejercicio 2.11 utilizando la librería: ieee.std_logic_l 164.all;



2.14 Declare la entidad del multiplexor de 4:1 mostrado en la figura del ejercicio 2.11, si cada entrada esta formada por un vector de 4 bits.

2.15 Declare la entidad del multiplicador mostrado en el ejercicio 2.2 utilizando vectores y el paquete std_logic_l 164.

Arquitecturas

2.16 Mediante un estilo funcional, programe en VHDL el funcionamiento de una lámpara para código Morse que encienda la luz al presionar un botón y la apague al soltarlo.

2.17 Con un estilo funcional, programe en VHDL el funcionamiento del motor de un ventilador en que el motor gire en un sentido al presionar el botón 'a' y en dirección contraria al oprimir el botón 'b\

2.18 Con un estilo por flujo de datos, programe en VHDL el funcionamiento de un panel en una fábrica de empaquetamiento de arroz. Este panel muestra el contenido de 2 silos (a, b) que tiene la fábrica para guardar el arroz; un sensor detecta cuán llenos están, cuando se encuentran al 100% de su capacidad, envía un '1 lógico', y cuando tienen 25% o menos envía un '0 lógico'; si en uno de estos silos disminuye el contenido a 25% o menos, se prende una luz (c), si los dos sobrepasan ese límite se enciende otra luz (d) y suena una alarma (e).

2.19 Mediante el estilo de programación por flujo de datos, programe en VHDL el funcionamiento de un robot en una planta que espera a que se llene una tarima con cuatro cajas antes de llevarla a la bodega de almacenamiento; para saber si la tarima está llena cuenta con cuatro sensores, cada uno apunta a sendas cajas; si hay una caja marca un '1 lógico'; si falta, marca un '0 lógico'. Si falta alguna caja el robot no se puede ir, cuando están las cuatro cajas el robot se lleva la tarima.

2.20 Con el estilo de programación por flujo de datos, programe en VHDL el funcionamiento de una caja de seguridad cuya apertura requiere la presión simultánea de tres de cuatro botones ('a', 'b\ 'c' y 'd'). Los botones que se deben oprimir son: 'a', 'c' y 'd'.

2.21 Mediante el estilo de programación estructural, programe en VHDL el problema del apagador de escalera. La función para este problema es $c - a\,b + ab$, donde *a* es el interruptor inferior, *b* es el interruptor superior y c es el foco.

2.22 Con un estilo estructural, programe en VHDL el funcionamiento de un motor que se enciende con la siguiente ecuación:

$$y = ab + cb + ac.$$

Bibliografía

Bergé J. M., Fonkua A., Maginot S.: *VHDL Designer's Reference.* Kluwer Academic Publisher, 1992.

Teres Ll., Torroja Y., Olcoz S., Villar E.: *VHDL Lenguaje Estándar de Diseño Electrónico.* McGraw-Hill, 1998.

Skahill K.: *VHDL for programmable logic.* Addison Wesley, 1996.

Mazor S., Laangstraar P: A *Guide to VHDL.* Kluwer Academic Publisher, 1993.

Maxinez G. David, Alcalá Jessica: *Diseño de Sistemas Embebidos a través del Lenguaje de Descripción en Hardware VHDL.* XIX Congreso Internacional Académico de Ingeniería Electrónica. México, 1997.

Kloos C., Cerny E.: *Hardware Description Language and their aplications. Specification, modelling, verification and synthesis of microelectronic systems.* Chapman & Hall, 1997.

IEEE: *The IEEE standard VHDL Language Reference Manual.* IEEE-Std-1076-1987, 1988.

Zainalabedin Navabi: *Analysis and Modeling of Digital Systems.* McGraw-Hill, 1988.

Otras lecturas sobre el tema

Armstrong James R., Gray F. G.: *Structured Logic Desing with VHDL.* Prentice Hall, 1993.

Capítulo 3

Diseño lógico combinacional mediante VHDL

Introducción

En este capítulo se diseñan los circuitos combinacionales más utilizados en el diseño lógico a través del lenguaje de descripción en hardware. Esto permite introducir nuevos conceptos, palabras reservadas, reglas, algoritmos, etc., que muestran la potencia y profundidad del lenguaje VHDL.

El desarrollo de cada una de las entidades de diseño descritas en este capítulo se puede optimizar mediante el uso adecuado de las declaraciones secuenciales, concurrentes o ambas, utilizando en esta descripción cualquiera de los tres tipos de arquitectura —funcional, por flujo de datos y estructural— vistos en el capítulo anterior. Sin embargo y dada la filosofía que queremos manejar en este texto, nos parece conveniente presentar soluciones que incluyan nuevas declaraciones, nuevos tipos de datos y nuevos algoritmos de análisis; es decir, no se pretende presentar la mejor opción de diseño para un problema; por el contrario, se propone brindar la mayor cantidad de soluciones (modelos) que le permitan deducir y construir sus estrategias de diseño para optimizar sus resultados.

3.1 Programación de estructuras básicas mediante declaraciones concurrentes

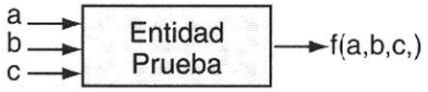
Como se mencionó antes, las declaraciones concurrentes se encuentran fuera de la declaración de un proceso y suelen usarse en las descripciones de flujo de datos y estructural. Esto se debe a que en una declaración concurrente no importa el orden en que se escriban las señales, ya que el resultado para determinada función sería el mismo.

En VHDL existen tres tipos de declaraciones concurrentes:

- Declaraciones condicionales asignadas a una señal (when-else)
- Declaraciones concurrentes asignadas a señales
- Selección de una señal (with-select-when)

3.1.1 Declaraciones condicionales asignadas a una señal (when'else)

La declaración when'else se utiliza para asignar valores a una señal, determinando así la ejecución de una condición propia del diseño. Para ejemplificar, consideremos la entidad mostrada en la figura 3.1, cuyo funcionamiento se define en la tabla de verdad.



a	b	c	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Figura 3.1 Declaraciones when-else.

La entidad se puede programar mediante *declaraciones condicionales* (when-else), debido a que este modelo permite definir paso a paso el comportamiento del sistema, según se muestra en el listado 3.1.

```
1 — Ejemplo combinacional básico
2 library ieee;
3 use ieee.std_logic_1164.all ;
4 entity tabla is port(
5       a,b,c: in std_logic;
6            f: out std_logic);
7 end tabla;
8 architecture ejemplo of tabla is
9 begin
10      £ <= '1' when (a='0' and b='0' and c='0') else
11           '1' when (a='0' and b='V' and c='1') else
12           'V' when (a='1' and b='1' and c='0') else
13           '1' when (a='1' and b='1' and c='1') else
14           '0';
15 end ejemplo;
```

Nótese que la función de salida f (línea 10) depende directamente de las condiciones que presentan las variables de entrada, además y dado que la ejecución inicial de una u otra condición no afecta la lógica del programa, el resultado es el mismo; es decir, la condición de entrada "111", visualizada en la tabla de verdad, puede ejecutarse antes que la condición "000" sin alterar el resultado final.

La ventaja de la programación en VHDL en comparación con el diseño lógico puede intuirse considerando que la función de salida f mediante álgebra booleana se representa con:

$$f = abc + abc + abc + abc$$

en el diseño convencional se utilizarían inversores, compuertas OR y compuertas AND; en VHDL la solución es directa utilizando la función lógica *and*. Como ejemplo, observemos que de la línea 10 a la 14 las instrucciones se interpretarían de la siguiente manera:

10 asigna a el valor de 1 cuando a = 0 y b = 0 y e = 0 si no
11 asigna a el valor de 1 cuando a = 0 y b = i y c = 1 si no
12 asigna a el valor de 1 cuando a = 1 y b - i y e - 0 si no
13 asigna a el valor de 1 cuando a = 0 y b = i y e = 1 si no
14 asigna a el valor de 0.

Operadores lógicos

Los operadores lógicos más utilizados en la descripción de funciones booleanas, y definidos en los diferentes tipos de datos bit, son los operadores `and, or, nand, xor, xnor y not`. Las operaciones que se efectúen entre ellos (excepto not) deben realizarse con datos que tengan la misma longitud o palabra de bits.

En el momento de ser compilados los operadores lógicos presentan el siguiente orden y prioridad:

1) Expresiones entre paréntesis
2) Complementos
3) Función AND
4) Función OR

Las operaciones `xor y xnor` son transparentes al compilador y las interpreta mediante la suma de productos correspondiente a su función.

Como ejemplo del uso de operadores lógicos en VHDL, observemos la siguiente comparación:

Ecuación	En VHDL
$q = a + x \bullet y$	q = a or (x *and* y)
$y = a + b*c + d$	y = *not* (a *or* (b *and not* c) *or* d)

Ejemplo 3.1 Una función F depende de cuatro variables D, C, B, A, que representan un número binario, donde A es la variable menos significativa. La función F adopta el valor de uno si el número formado por las cuatro variables es inferior o igual a 7 y superior a 3. En caso contrario la función F es cero.

- Obtenga la tabla de verdad de la función F y realice el programa correspondiente en VHDL (utilice estructuras del tipo when-else y operadores lógicos).

Solución

Primero analizamos el enunciado y estructuramos la siguiente tabla de verdad según las especificaciones indicadas.

D	C	B	A	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

A partir de la tabla anterior, se puede programar la función F utilizando declaraciones condicionales when-else. El código VHDL es el siguiente:

```
1 library ieee;
2 vise ieee. std_logic_1164. all ;
3 entity función is port (
4    D,C,B,A: in std_logic;
5    F: out std_logic);
6 end función;
7 architecture a_func of función is
8 begin
9    F <=  '1' when (A = '0' and B = '0' and C = •1' and D = '0' ) else
10         '1' when (A = '1' and B = '0' and C = •1' and D = '0' ) else
11         '1' when (A = '0' and B = '1' and c = •V and D = '0' ) else
12         '1' when (A = •V and B = '1' and c = •1' and D = '0') else
13         '0';
14 end a_func;
```

3.1.2 Declaraciones concurrentes asignadas a señales

En este tipo de declaración encontraremos las funciones de salida mediante la ecuación booleana que describe el comportamiento de cada una de las compuertas. Obsérvese que ahora el circuito de la figura 3.2 cuenta con tres salidas (x1, x2 y x3) en lugar de una.

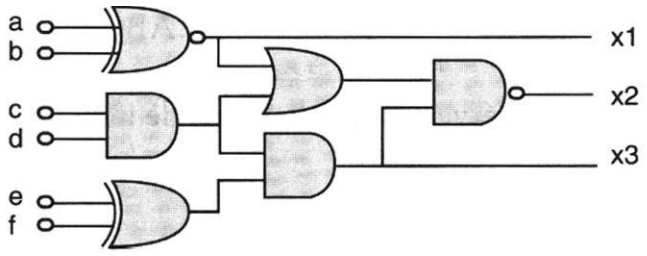


Figura 3.2 Circuito lógico realizado con compuertas.

El programa correspondiente al circuito de la figura 3.2 se muestra en el listado 3.2.

```
library ieee;
use ieee.std_logic_1164.all;
entity logic is port (
 a,b,c,d,e,f: in std_logic;
 xl,x2,x3:    out std_logic) ;
end logic;
architecture booleana of logic is
begin
      xl <= a xnor b;
      x2 <= ( ( (c and d)or(a xnor b) ) nand
             ( (e xor f)and(c and d) ) ) ;
      x3 <= (e xnor f) and (c and d) ;
end booleana;
```

Listado 3.1 Declaraciones concurrentes asignadas a señales.

Ejemplo 3.2 Dada la tabla de verdad mostrada a continuación, halle las ecuaciones X, Y, Z, de la forma suma de productos y prográmelas en VHDL, utilizando declaraciones concurrentes asignadas a señales.

A	B	C	X	Y	z
0	0	0	1	0	1
0	0	1	1	1	0
0	1	0	0	0	1
0	1	1	1	0	1
1	0	0	0	0	0
1	0	1	0	1	0
1	1	0	0	1	0
1	1	1	1	0	0

Solución

Las ecuaciones de la forma suma de productos para X, Y y Z se muestran a continuación:

1) X = ABC + ÁBC + ÀBC + ABC
2) Y = ÁBC + ABC + ABC
3) Z = ÁBC + ABC + ÁBC

Obsérvese ahora la forma de implementar estas ecuaciones por medio de operadores lógicos.

```
1 library ieee;
2 use ieee.std_logic_1164.all ;
3 entity concurrente is port (
4       A,B,C: in std_logic;
5       X,Y,Z: out std_logic);
6 end concurrente;
7 architecture a_conc of concurrente is
8 begin
9          X <= (not A and not B and not C) or (not A and not B and C)
10              or (not a and B and C ) or (A and B and C) ;
11         Y <= (not A and not B and C) or (A and not B and C)
12              or (A and B and not C) ;
13         Z <= (not A and not B and not C) or (not A and B and not C)
14              or (not A and B and C) ;
15 end a_conc;
```

3.1.3 Selección de una señal (with*select>when)

La declaración `with'Select-when` se utiliza para asignar un valor a una señal con base en el valor de otra señal previamente seleccionada. Por ejemplo, en el listado correspondiente a la figura 3.3 se muestra el código que representa a este tipo de declaración. Como puede observarse, el valor de la salida C depende de las señales de entrada seleccionadas a(0) y a(1), de acuerdo con la tabla de verdad correspondiente.

a(0)	a(1)	c
0	0	1
0	1	0
1	0	1
1	1	0

Figura 3.3 Tabla de verdad.

```
library ieee;
use ieee.std_logic_1164.all;
entity circuito is port(
     a: in std_logic_vector (1 downto 0) ;
     c: out std_logic);
end circuito;
architecture arq_cir of circuito is
begin
     with a select
       c <= '1' when "00",
            '0' when "01",
            '1' when "10",
            '0' when others[1];
end arq_cir;
```

Listado 3.3 Código VHDL correspondiente a la tabla de verdad de la figura 3.3.

Ejemplo 3.3 Se requiere diseñar un circuito combinacional que detecte números primos de 4 bits. Realice la tabla de verdad v elabore un programa que describa su función. Utilice instrucciones del tipo with - select - when.

[1] El uso de la palabra reservada others se explica a detalle en la sección multiplexores de este capítulo.

Solución

La tabla de verdad que resuelve la función es la siguiente:

XO	XI	X2	X3	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Si se considera que la entrada X es un vector de 4 bits y que F es la función de salida, el programa en VHDL quedaría de la siguiente manera.

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity selección is port (
4       X: in std_logic_vector(0 to 3);
5       F: out std_logic) ;
6 end selección;
7 architecture a_selec of selección is
8 begin
9          with X select
10         F <= '1' when "0001",
11                 '1' when "0010",
12                 '1' when "0011",
13                 '1' when "0101",
14                 '1' when "0111",
15                 '1' when "1011",
16                 '1' when "1101",
17                 '0' when others;
18 end a_selec;
```

3.2 Programación de estructuras básicas mediante declaraciones secuenciales

Como ya se mencionó, las declaraciones secuenciales son aquellas en las que el orden que llevan puede tener un efecto significativo en la lógica descrita. A diferencia de una declaración concurrente, una secuencial debe ejecutarse en el orden en que aparece y formar parte de un proceso (process).

Declaración if-then-else (si-entonces-si no). Esta declaración sirve para seleccionar una condición o condiciones basadas en el resultado de evaluaciones lógicas (falso o verdadero). Por ejemplo, observemos que en la instrucción:

```
if la condición es cierta then
        realiza la operación 1;
else
        realiza la operación 2;
end if;
```

si (if) *condición* se evalúa como verdadera, entonces (then) la instrucción indica que se ejecutará la *operación 1.* Por el contrario, si la condición se evalúa como falsa (else) correrá la *operación* 2. La instrucción que indica el fin de la declaración es end if (fin del si). Un ejemplo que ilustra este tipo de declaración se encuentra en la figura 2.6 y por comodidad se repite en la figura 3.4.

Figura 3.4 Comparador de igualdad de dos bits.

El código correspondiente a esta entidad de diseño se muestra en el listado 3.4.

```
1  —Ejemplo de declaración de la entidad comparador
2       entity comp is
3       port  (a,b: in bit_vector( 1 downto 0);
4                c: out bit) ;
5       end comp ;
6       architecture funcional of comp is
7       begin
8       compara: process  (a,b)
9       begin
10           if a = b then
11                c  <='1';
12           else
13                c <= '0';
14      end if;
15      end process compara ;
16      end  funcional;
```

Listado 3.4 Declaración secuencial de un comparador de igualdad de dos bits.

Notemos cómo en este tipo de ejemplos sólo son necesarias dos condiciones por evaluar, pero no en todos los diseños es así. Por tanto, cuando se requieren más condiciones de control, se utiliza una nueva estructura llamada elsif (si no-si), la cual permite expandir y especificar prioridades dentro del proceso. La sintaxis para esta operación es:

if *la condición 1 se cumple* then
  realiza operación 1;
elsif *la condición 2 se cumple* then
  realiza operación 2:
else
  realiza operación 3;
end if;

la cual se interpreta como sigue: Si (if) la *condición 1* es verdadera, entonces (then) se ejecuta la *operación l,* si no-si (elsif) se evalúa la *condición* 2 y si es verdadera entonces (then) se ejecuta la *operación 2,* si no (else) se ejecuta la *operación 3.*

3.2.1 Comparador de magnitud de 4 bits

La forma de utilizar las declaraciones secuenciales se ilustra en el diseño de un comparador de dos números de 4 bits, figura 3.5a). En este caso el sistema tiene tres salidas que indican cuando uno de los números es mayor, igual

o menor que el otro. La figura 3.5b) representa el mismo comparador de manera simplificada utilizando la notación vectorial.

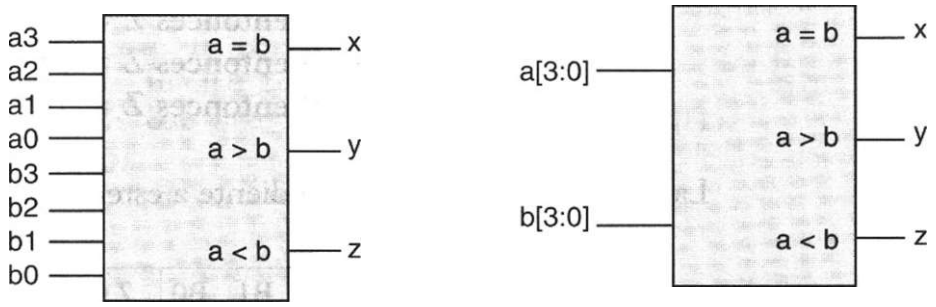


Figura 3.5 a) Comparador de 4 bits, b) Comparador expresado con vectores de 4 bits.

En el listado 3.5 se observa el algoritmo en VHDL que describe el funcionamiento del comparador. En la línea 9 se muestra la lista sensitiva (a y b) del proceso (process). En las líneas 10 a 17 el proceso se desenvuelve mediante el análisis de las variables de la lista sensitiva. Sin mucho esfuerzo puede verse que si a = b, entonces x toma el valor de 1, de forma similar se intuye el comportamiento para a > b y a < b, incluyendo la declaración elsif.

```
1 library ieee;
2 use ieee.std_logic_1164.all ;
3 entity comp4 is port (
4 a,b:   in std_logic_vector(3 downto 0);
5 x,y,z: out std_logic);
6 end comp4;
7 architecture arq_comp4 of comp4 is
8 begin
9    process  (a,b)
10   begin
11          if  (a = b)  then
12            x <= '1';
13          elsif  (a > b)  then
14            y <= '1';
15          else
16             z <= '1';
17       end if;
18       end process;
19 end arq_comp4 ;
```

Listado 3.5 Descripción del comparador de 4 bits utilizando el estilo funcional.

Ejemplo 3.4 Diseñe un comparador de dos números A y B, cada número formado por dos bits (Al AO) y (Bl BO) la salida del comparador también es de dos bits y está representada por la variable Z (Z1 ZO) de tal forma que si:

A = B entonces Z = 11
A < B entonces Z = 01
A > B entonces Z = 10

La tabla de verdad correspondiente a este comparador es la siguiente.

A1	AO	B1	BO	Z1	ZO
0	0	0	0	1	1
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	1	1
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	1	1

Las ecuaciones lógicas reducidas mediante un mapa de Karnaugh para Z1 y ZO son las siguientes:

$$Z1 = AO\ A1 + BO\ B1 + A1\ BO + AO\ BO + AO\ B1$$
$$ZO = \acute{A}O\ A1 + BO\ B1 + \acute{A}O\ B1 + \acute{A}O\ B0 + A1\ BO$$

El circuito representativo de este comparador es el siguiente, figura E3.4.

Figura E3.4 Descripción funcional de un comparador de igualdad de dos bits.

La programación de este comparador se muestra en el siguiente listado.

```
library ieee;
use ieee.std_logic_1164.all;
entity comp is port (
    A,B: in std_logic_vector(1 downto 0);
      Z: out std_logic_vector (1 downto 0));
end comp;
architecture a_comp of comp is
begin
   process (A,B) begin

        if A = B then

          Z <= "11";

       elsif A < B then

          Z <= "01";

       else

          Z <= "10";
       end if;
    end process;
 end a_comp;
```

Operadores relaciónales. Los operadores relaciónales se usan para evaluar la igualdad, desigualdad o la magnitud en una expresión. Los operadores de igualdad y desigualdad (= y /=) se definen en todos los tipos de datos. Los operadores de magnitud (<,<—,> y >=) lo están sólo dentro del tipo escalar. En ambos casos debe considerarse que el tamaño de los vectores en que se aplicarán dichos operadores debe ser igual. En la tabla 3.1 se muestran estos operadores y su significado.

Operador	Significado
=	Igual
/=	Diferente
<	Menor
< =	Menor o igual
>	Mayor
> =	Mayor o igual

Tabla 3.1 Operadores relaciónales.

3.2.2 Buffers triestado

Los registros de tres estados (buffers tri-estado) tienen diversas aplicaciones, ya sea como salidas de sistemas (modo buffer) o como parte integral de un circuito. En VHDL estos dispositivos son definidos a través de los valores que manejan (0,1 y alta impedancia 'Z'). En la figura 3.6 se observa el diagrama correspondiente a este circuito, y en el listado 3.6 el código que describe su funcionamiento.

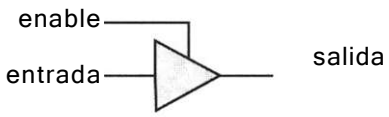


Figura 3.6 Buffer tri-estado.

```
library ieee;
use ieee.std_logic_1164.all ;
entity tri_est is port(
 enable, entrada: in std_logic;
           salida: out std_logic);
end tri_est;
architecture arq_buffer of tri_est is
begin
   process (enable, entrada) begin
       if enable = '0' then
          salida <= 'Z';
       else
          salida <= entrada;
       end if;
   end process ;
end arq_buffer;
```

Listado 3.6 Descripción mediante valores de alta impedancia.

El listado anterior se basa en un proceso, el cual se utiliza para describir los valores que tomará la salida del registro (buffer). En este proceso se indica que cuando se confirma el habilitador del circuito (*enable*), el valor que se

encuentra a la entrada del circuito se asigna a la salida; si por el contrario no se confirma *enable,* la salida del buffer tomará un valor de alta impedancia (Z). El tipo std_logic soporta este valor —al igual que 0 y 1—. A esto se debe que en el diseño se prefiera utilizar el estándar std_logic_l 164 y no el tipo bit, ya que el primero es más versátil al proveer valores de *alta impedancia* y condiciones de *no importa* (-), los cuales no están considerados en el tipo bit.

3.2.3 Multiplexores

Los multiplexores se diseñan describiendo su comportamiento mediante la declaración with-select'when o ecuaciones booleanas.

En la figura 3.7a) se observa que el multiplexor dual tiene como entrada de datos las variables a, b, c y d, cada una de ellas representadas por dos bits (a1, a0), (bl, bO), etc., las líneas de selección (s) de dos bits (si y sO) y la línea de salida z (zl y zO).

En la figura 3.7b) se muestra un diagrama simplificado que resalta la representación mediante vectores de bits.

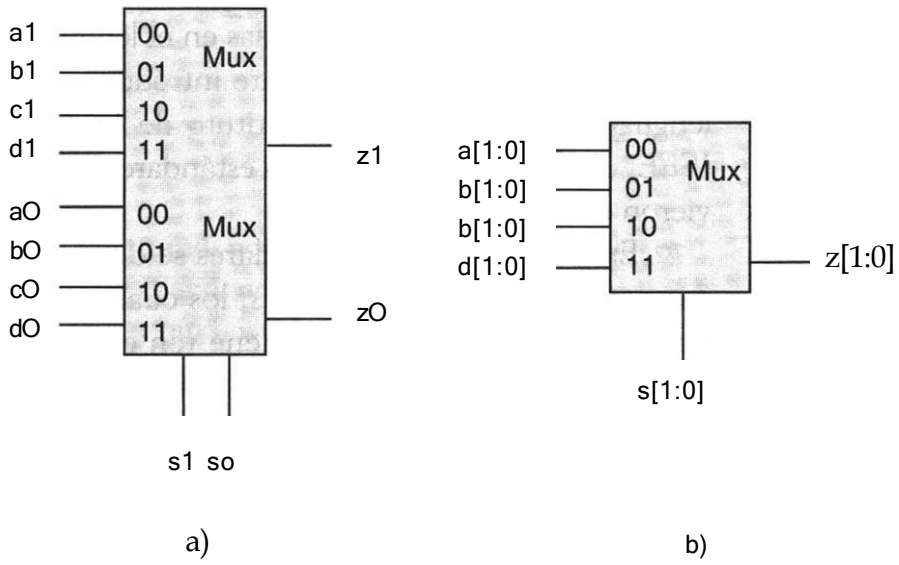


Figura 3.7 a) Multiplexor de 4 bits, **b)** Multiplexor con vectores.

En el listado 3.7 se muestra la descripción mediante with-select-when del multiplexor dual de 2 x 4. En este caso la señal s determina cuál de las cuatro señales se asigna a la salida Z. Los valores de s están dados como "00", "01" y "10"; el término others (otros) especifica cualquier combinación adicional que pudiera presentarse (que incluye el "11"), ya que esta variable se encuentra definida dentro del tipo std_logic_vector, el cual contiene nueve

valores posibles que la herramienta de síntesis[2] reconoce como tipos lógicos estándares.

```
library ieee;
use ieee.std_logic_1164.all ;
entity mux is port(
 a,b,c,d: in std_logic_vector(1 downto 0);
        s: in std_logic_vector (1 downto 0) ;
        Z: out std_logic_vector (1 downto 0) ;
end mux;

architecture arqmux4 of mux is
begin
with s select
 Z <=   a when "00",
        b when "01",
        c when "10",
        d when others ;
end arqmux4;
```

Listado 3.7 Multiplexor descrito con declaraciones **with-select-when.**

Tipos lógicos estándares

Las funciones estándares definidas en el lenguaje VHDL se crearon para evitar que cada distribuidor de software introdujera sus paquetes y tipos de datos al lenguaje. Por esta razón el Instituto de Ingenieros Eléctricos y Electrónicos, *IEEE,* estableció desde 1987 los estándares *stdjogíc* y *stdjogicjuector,* que ya se vieron en un capítulo anterior.

En cada uno de los estándares se definen ciertos tipos de datos conocidos como *tipos lógicos estándares,* los cuales se pueden utilizar haciendo referencia al paquete que los contiene (en este caso *std_logic_1164).* En la tabla 3.2 se muestran los tipos lógicos estándares definidos en VHDL.

V'	-	**Valor no inicializado**
'X'	-	**Valor fuerte desconocido**
'0'	-	**0 Fuerte**
T	-	**1 Fuerte**
T	-	**Alta impedancia**
'W'	-	**Valor débil desconocido**
'C	-	**0 débil**
'H'	-	**1 débil**
	-	**No importa (don't care));**

Tabla 3.2 Tipos lógicos estándares definidos en VHDL.

[2] La herramienta de síntesis es el software incluido en VHDL, que genera las ecuaciones de diseño de cualquier circuito. Esta herramienta permite implementar diseños sin el formato de ecuaciones booleaneas que utilizan otros programas.

Como se puede apreciar, estos tipos de datos no tienen un significado evidente para el diseñador, pero sí lo tienen en la compilación del programa. Por ejemplo, el estándar no especifica alguna interpretación de L y H, debido a que la mayoría de las herramientas no los soportan. Los valores metalógicos[3] ('U', 'W', 'X','-') carecen de sentido en la síntesis, pero la herramienta los usa en la simulación del código. Como se mencionó en la sección anterior, el uso de 'Z' entraña un valor de alta impedancia.

3.2.4 Descripción de multiplexores mediante ecuaciones booleanas

En el listado 3.8 se muestra una solución con ecuaciones booleanas para el multiplexor dual de la figura 3.7.

```
library ieee;
use ieee.std_logic_1164.all;
entity mux is port (
 a,b,c,d: in std_logic_vector(1 downto 0);
 s:       in std_logic_vector(1 downto 0) ;
 z:       out std_logic_vector(1 downto 0));
end mux;

architecture arqmux of mux is
begin

 z(1) <=(a(1) and not(s(1)) and not(s(0))) or
        (b(1) and not(s(1)) and s(0)) or
        (c(1) and s(1) and not (s(0))) or
        (d(1) and s(1) and s(0)) ;

 z(0) <= (a(0) and not (s(1)) and not (s(0))) or
        (b(0) and not(s(1)) and s(0)) or
        (c(0) and s(1) and not(s(0))) or
        (d(0) and s(1) and s(0));

end arqmux;
```

Listado 3.8 Multiplexor descrito con ecuaciones booleanas.

[3] Un dato metalógico consiste en los valores definidos para el tipo std_logic. los cuales están contenidos en el paquete estándar IEEE 1164.

3.2.5 Sumadores

Diseño de un circuito medio sumador

Para describir el funcionamiento de los circuitos sumadores es importante recordar las reglas básicas de la adición.

$$0 + 0 = 0$$
$$0+1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 10$$

En el caso de la suma 1 + 1 = 10, el resultado "0" representa el valor de la suma, mientras que el "1" el valor del acarreo.

Para observar en detalle el funcionamiento de un circuito medio sumador, considere la suma de los números A y B mostrados en la tabla 3.3.

A	B	Suma	Cout	
0	0	0	0	
0	1	1	0	
1	0	1	0	
1	1	0	1	(1 + 1 = 10)

Tabla 3.3 Tabla de verdad de un circuito sumador.

La ecuación lógica que corresponde a la expresión *Suma* = $A\,B$ + A B es la función lógica or-exclusiva A © B, mientras que la ecuación lógica del acarreo de salida es *Cout* = AB que corresponde a la compuerta lógica *and.* La realización física de estas ecuaciones se muestra en la figura 3.8a, en ella se presenta el bloque lógico del medio sumador (MS) y la figura 3.8b representa su implantación mediante compuertas lógicas.

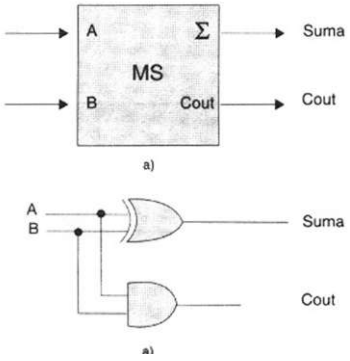


Figura 3.8 a) Diagrama a bloques de un medio sumador; **b)** Medio sumador lógico.

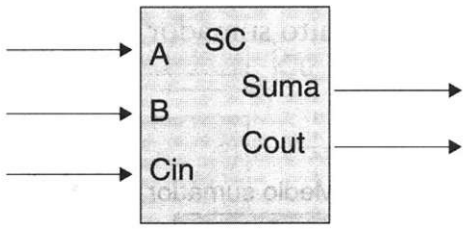
El programa en VHDL que representa este medio sumador se muestra en el listado 3.9.

```
library ieee;
use ieee.std_logic_1164.all ;
entity m_sum is port  (
 A,B: in std_logic;
 SUMA, Cout: out std_logic) ;
end m_sum;
architecture am_sum of m_sum is
begin
    SUMA <= A XOR B;
    Cout <= A AND B;
end am_sum;
```

Listado 3.9 Código de un medio sumador.

Diseño de un sumador completo

Un sumador completo (SC) a diferencia del circuito medio sumador considera un acarreo de entrada (Cin) tal y como se muestra en la figura 3.9a, el comportamiento de este sumador se describe a través de su tabla de verdad.



A	B	Cin	Suma	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Listado 3.9 a) Sumador completo; b) Tabla de verdad del medio sumador.

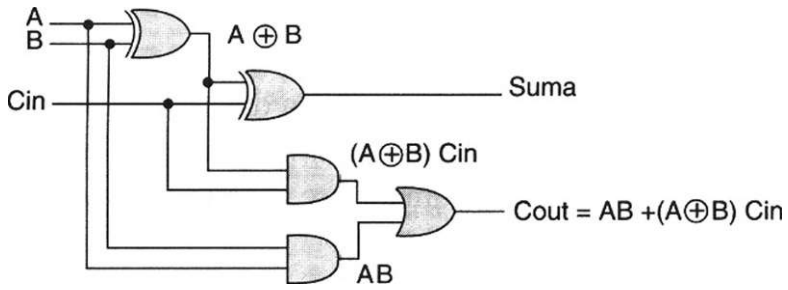
Las ecuaciones reducidas mediante un mapa de Karnaugh correspondientes a la salida Suma y Cout se muestran a continuación

Suma = Á B Cin + Á B Cin + A B C i ñ + A B C i n
Cout = A B +B Cin + A Cin

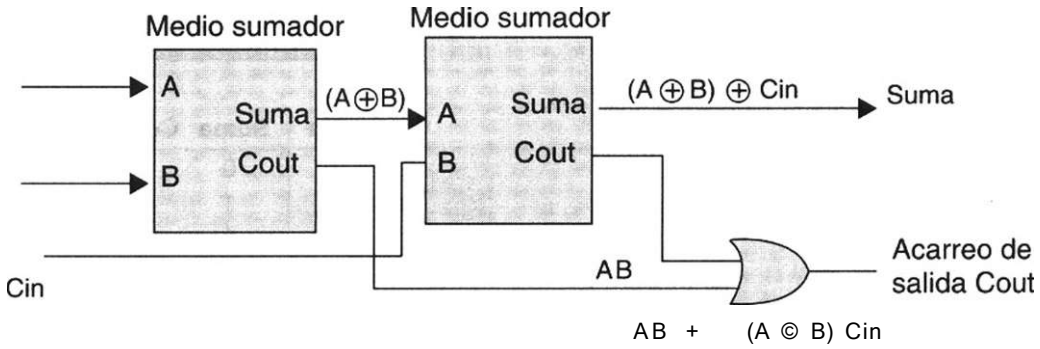
Si se manipulan las ecuaciones anteriores mediante álgebra booleana obtenemos que la función de Suma y Cout puede expresarse como:

Suma = A© B ©C i n
Cout = AB + (A © B) Cin

La realización física del circuito se basa en la utilización de compuertas or-exclusiva como se muestra en la figura 3.10 a). Como puede observarse en la figura 3.10 b), dos circuitos medio sumadores pueden implementar un sumador completo.



a) Circuito sumador completo implementado por compuertas.



b) Circuito sumador completo implementado por medio sumadores

Figura 3.10 Circuitos sumadores.

La programación en VHDL del sumador completo se presenta en el listado 3.10.

```
library ieee;
use ieee.std_logic_1164.all;
entity sum is port (
 A,B,Cin: in std_logic;
 Suma, Cout: out std_logic);
end sum;
architecture a_sum of sum is
begin
      Suma <= A xor B xor Cin;
      Cout <= (A and B) or (A xor B) and Cin;
end a_sum;
```

Listado 3.10 Sumador Completo.

Sumador Paralelo de 4 bits

Según lo anterior, para realizar un sumador paralelo de 4 bits sólo se requiere conectar en cascada un circuito medio sumador y tres sumadores completos como se muestra en la figura 3.11

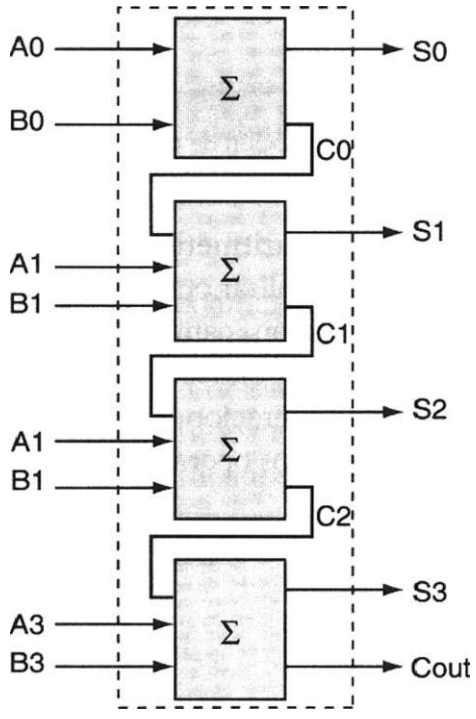


Figura 3.11 Sumador de 4 bits.

A su vez, éste es un buen ejemplo para reafirmar el manejo de señales (signal). En la figura 3.11 se observa cómo los acarreos de salida (C0, C1 y C2) se encuentran retroalimentados dentro del circuito, por lo que no tienen un pin externo asignado. En el listado 3.11 se ilustra la programación en VHDL.

Como puede apreciarse, el intervalo utilizado dentro de signal es (0 to 2), debido a que sólo se retroalimentan los acarreos C0, C1 y C2. Por otro lado, con un poco de esfuerzo puede intuirse que cada uno de los diferentes bloques que forma el sumador se caracteriza usando compuertas *xor* (or exclusiva).

```
library ieee;
use ieee.std_logic_1164.all;
entity suma is port(
 A,B: in std_logic_vector (0 to 3) ;
   S: out std_logic_vector {0 to 3);
   Cout: out std_logic);
end suma ;
architecture arqsuma of suma is
 signal C: std_logic_vector(0 to 2);
  begin
        S(0) <= A(0) xor B(0);
        C(0) <= A(0) and B(0);
        S(1) <= (A(1) xor B(1)) xor C(0);
        C(1) <= (A(1) and B(1) ) or (C(0)and(A(l)xor B(l)));
        S (2) <= (A (2) xor B (2) ) xor C(1);
        C (2) <= (A (2) and B(2)) or (C (1) and (A(2) xor B(2)));
        S (3) <= (A(3) xor B(3)) xor C(2);
        Cout <= (A(3) and B(3)) or (C(2)and(A(3)xor B(3)));
end arqsuma ;
```

Listado 3.11 Descripción de un sumador de 4 bits.

Operadores aritméticos. Como su nombre indica, los operadores aritméticos permiten realizar operaciones del tipo aritmético, como suma, resta, multiplicación, división, cambios de signo, valor absoluto y concatenación. Estos operadores suelen usarse en el diseño lógico para describir sumadores y restadores o en las operaciones de incremento y decremento de datos. En la tabla 3.4 se muestran los operadores aritméticos predefinidos en VHDL.

Operador	Descripción
+	Suma
-	Resta
/	División
*	Multiplicación
**	Potencia

Tabla 3.4 Operadores aritméticos utilizados en VHDL

Como ejemplo, analicemos el diseño de un circuito sumador de 4 bits que no considera el acarreo de salida (listado 3.12).

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity sum is port (
 A,B:  in std_logic_vector(0 to 3);
 Suma: out std_logic_vector(0 to 3));
end sum;
architecture arqsum of sum is
begin
 Suma <= A + B;
end arqsum;
```

Listado 3.12 Descripción de un sumador mediante std_logic_vector y el paquete std arith.

En el listado 3.12 se introdujo el paquete *std_arith,* el cual — como ya se mencionó — se encuentra en la librería de trabajo *work.* Este paquete permite el uso de los operadores aritméticos con operaciones realizadas entre arreglos del tipo *std_logic_vector;* es decir, dado que dentro del paquete estándar *(std_logic_l 164)* no están definidos los operadores aritméticos, es necesario usar el paquete std_arith.

El uso de los operadores existentes en VHDL, así como los tipos de datos para los cuales se encuentran definidos, se incluye en el apéndice B.

3.2.6 Decodificadores

La programación de circuitos decodificadores se basa en el uso de declaraciones que permiten establecer la relación entre un código binario aplicado a las entradas del dispositivo y el nivel de salida obtenido.

En esta sección se presentan dos tipos de decodificadores: el decodificador BCD-decimal y el decodificador de BCD a siete segmentos, ya que consideramos que son dos de los más utilizados en el diseño lógico combinacional.

Decodificador BCD a decimal

En la figura 3.12 podemos observar la entidad de diseño correspondiente a un circuito decodificador, el cual convierte código BCD (código de binario a decimal) en uno de los diez dígitos decimales. Por lo general estos dispositivos

se conocen como decodificadores de 4 a 10 líneas, ya que contienen cuatro líneas de entrada y 10 de salida.

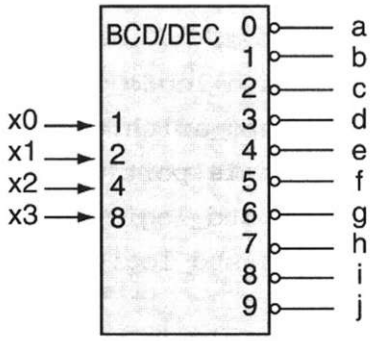


Figura 3.12 Decodificador de BCD a decimal.

El programa que describe el comportamiento de la entidad de la figura 3.12 se encuentra en el listado 3.13.

Como se puede apreciar, el código correspondiente a este circuito se basa en la ejecución de un proceso en que se establecen las condiciones que se evalúan para activar cada salida de acuerdo con el valor binario correspondiente. Para fines prácticos se asignó a cada salida un nombre a fin de facilitar su identificación.

A manera de ejemplo consideremos el valor de la entrada x = 0010, la cual corresponde al dígito decimal 2. Nótese cómo la condición que determina la asignación del valor evalúa primero la condición de x y si la confirma, asigna a la salida c el valor correspondiente al dígito decimal 2.

Por otro lado, se puede ver que al inicio del proceso se declararon todas las salidas con un valor inicial de '1', esto fue con el fin de asegurar que permanecieran desactivadas cuando no estuvieran en evaluación.

```
—Decodificador de BCD a decimal
library ieee;
use ieee.std_logic_1164.all;
entity deco is port (
 x: in std_logic_vector(3 downto 0);
      a,b,c,d,e,f, g,h,i,j: out std_logic);
end deco;
architecture arqdeco of deco is
begin

   process (x) begin
      a <= '1';
      b <= '1';
      c <= •!';
      d <= '!';
```

Continúa

```
    e <= '1';
    f <= '1';
    g <= '1';
    h <= '1';
    i <= '1';
    j <= '1';

   if X = "0000" then
        a <= '0';
  elsif X = "0001" then
        b <= '0';
  elsif X = "0010" then
        c <= '0';
  elsif X = "0011" then
        d <= '0';
  elsif X = "0100" then
        e <= '0';
  elsif X = "0101" then
        f <= '0';
  elsif X = "0110" then
        g <= '0';
  elsif X = "0111" then
        h <= '0';
  elsif X = "1000" then
        i <= '0';
  else
        j <= '0';
   end if;
  end process;
end arqdeco;
```

Listado 3.13 Descripción de un decodificador de BCD a decimai.

Decodificador de BCD a display de siete segmentos

En la figura 3.13a) se muestra un circuito decodificador, el cual acepta código BCD en sus entradas y proporciona salidas capaces de excitar un display de siete segmentos que indica el dígito decimal seleccionado. En la figura 3.13b) se observa la distribución de los segmentos dentro del display.

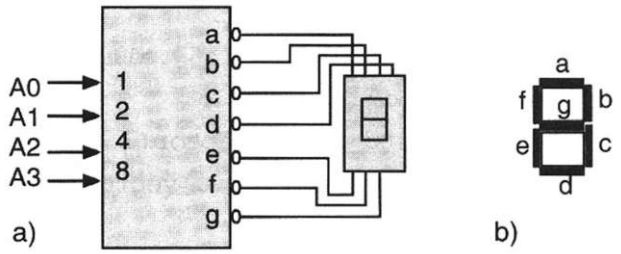


Figura 3.13 a) Decodificador BCD a siete segmentos, **b)** Configuración del display de siete segmentos.

Como se puede apreciar, la entidad del decodificador cuenta con una entrada llamada A, formada por cuatro bits (AO, A1, A2, A3), y siete salidas (a, b, c, d, e, f, g) activas en nivel bajo, las cuales corresponden a los segmentos del display. En la tabla 3.5 se indican los valores lógicos de salida correspondientes a cada segmento.

"A" Código BCD				Segmento del display "d"						
AO	A1	A2	A3	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0
0	0	1	1	0	0	0	0	1	1	0
0	1	0	0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	0	1	0	0
0	1	1	0	0	1	0	0	0	0	0
0	1	1	1	0	0	0	1	1	1	0
1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1	0	0

Tabla 3.5 Valores lógicos correspondientes a cada segmento del display.

La función del programa cuyo código se exhibe en el listado 3.14 utiliza declaraciones secuenciales del tipo case-when que, como se puede apreciar, ejecutan un conjunto de instrucciones basadas en el valor que pueda tomar una señal. En nuestro ejemplo, se describe de qué manera se maneja el decodificador de acuerdo con el valor que toma la señal A. Para fines prácticos se declararon todas las salidas como un solo vector de bits (identificado como d); de esta forma se entiende que la salida *a* corresponde al valor *dO,* la *b* al valor *di,* etc. Por otro lado, la palabra reservada others, como ya se indicó, define los valores metalógicos que puede tomar en la síntesis la salida *d.*

```
library ieee;
use ieee.std_logic_1164.all ;
entity deco is port (
  A: in std_logic_vector(3 downto 0);
     d: out std_logic_vector(6 downto 0));
end deco;
architecture arqdeco of deco is
begin
```

Continúa

```
    process (A) begin
     case A is
      when "0000" => d <= "0000001";
      when "0001" => d <= "1001111";
      when "0010" => d <= "0010010";
      when "0011" => d <= "0000110";
      when "0100" => d <= "1001100";
      when "0101"    d    "0100100";
      when "0110" => d <= "0100000";
      when "0111" => d <= "0001110";
      when "1000" => d <= "0000000";
      when "1001" => d <= "0000100";
      when others => d <= "1111111";
     end case;
   end process;
end arqdeco;
```

Listado 3.14 Uso de declaraciones case-when.

En la instrucción case-when podemos observar el uso de *asignaciones dobles (*=> d < =), las cuales permiten que una señal adopte un determinado valor de acuerdo con el cumplimiento de una condición especificada. Por ejemplo, en nuestro código (listado 3.14) estas instrucciones se interpretan de la siguiente manera: cuando la señal A sea "0000", asigna a la señal *d* el valor "0000001"; cuando A - "0001", asigna a *d* el valor "1001111", etc. Cabe mencionar que en la simulación del programa esta asignación se realiza simultánea.

3.2.7 Codificadores

En esta sección se presenta la forma de programar un circuito codificador, el cual como se observa en la figura 3.14, posee 10 entradas (cada una correspondiente a un dígito decimal) y cuatro salidas para el código binario de 4 bits BCD.

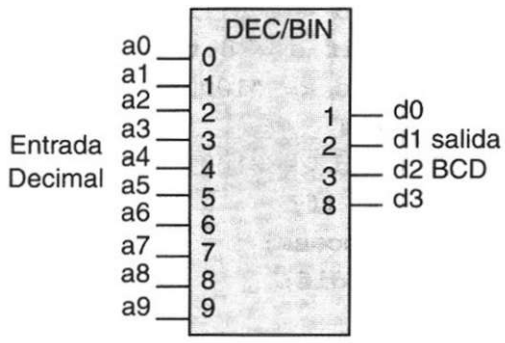


Figura 3.14 Codificador de decimal a BCD.

El programa que describe al circuito de la figura 3.14, se muestra en el listado 3.15. Como se puede ver, el puerto de entrada" a se declara como un vector para indicar la numeración decimal del 0 al 9, mientras que la salida binaria se realiza a través del vector "d". Asimismo, observe que la programación se realizó utilizando declaraciones del tipo if-then-elsif.

```
— Codificador de decimal a BCD

library ieee;
use ieee.std_logic_1164.all ;
entity codif is port (
  a: in integer range 0 to 9;
     d: out std_logic_vector(3 downto 0));
end codif;

architecture arqcodif of codif is
begin

   process (a)
     begin
        If a = 0 then
        d <= "0000";
     elsif a = 1 then
        d <= "0001";
     elsif a = 2 then
        d <= "0010";
     elsif a = 3 then
        d <= "0011";
     elsif a = 4 then
        d <= "0100";
     elsif a = 5 then
        d <= "0101";
     elsif a = 6 then
        d <= "0110";
     elsif a = 7 then
        d <= "0111";
     elsif a = 8 then
        d <= "1000";
     else
        d <= "1001";
     end if;
  end process;
end arqcodif;
```

Listado 3.15 Diseño de un codificador de decimal a binario.

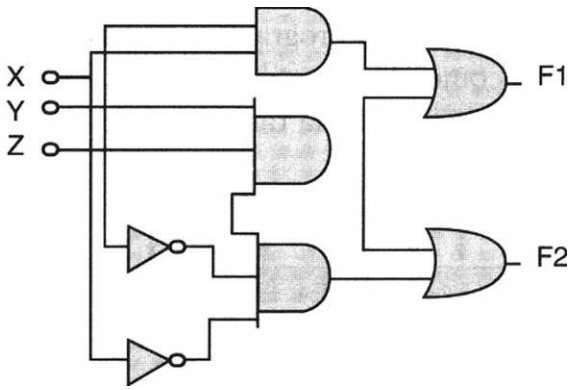
Ejercicios

Declaraciones concurrentes

3.1 Mencione los tres tipos de declaraciones concurrentes en VHDL.
3.2 Indique qué tipo de instrucciones se usan en las declaraciones condicionales asignadas a una señal.
3.3 Dé el orden de ejecución de los operadores lógicos en VHDL.
3.4 En las siguientes expresiones proporcione la expresión equivalente en VHDL.

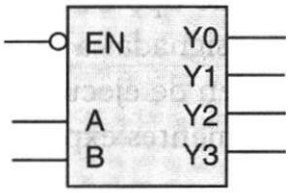
a) X = (a + b) (c *xor* d)
b) F = (a + c + d) + (a • d • c) • (a + b)
c) Z = (wx»y) + (x *xnor* y)

3.5 Mencione la principal diferencia entre las declaraciones secuenciales y las declaraciones concurrentes.
3.6 Mencione qué tipo de instrucciones utilizan las declaraciones secuenciales.
3.7 Indique qué instrucción se utiliza cuando se requieren más condiciones de evaluación en un proceso.
3.8 Mencione los operadores aritméticos que se utilizan en VHDL.
3.9 Indique en cuál librería y en qué paquete se encuentran los operadores de la pregunta 3.8.
3.10 Un circuito comparador de 3 bits recibe dos números de 3 bits X — X2, XI, XO y Z — Z2, Zl, ZO. Diseñe un programa en VHDL que produzca una salida F = 1 si y sólo si X < Z.
3.11 Elabore un programa en VHDL que describa el funcionamiento del circuito mostrado en la figura siguiente.



Decodificadores

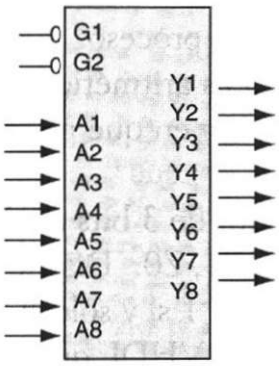
3.12 Se requiere un programa en VHDL de un circuito decodificador de 2 a 4, según se muestra en el siguiente diagrama. Utilice estructuras del tino if**^then**-elsif.



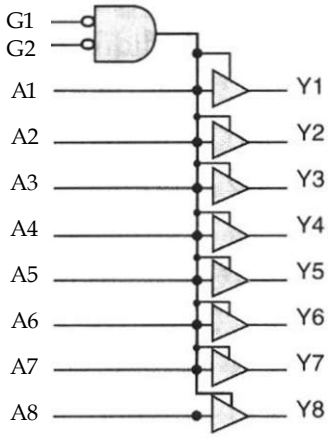
Donde:

EN — Entrada de habilitación del circuito (se activa en baja)
A, B = Entradas del circuito
Y[0:3] = Salidas del circuito

3.13 Con base en el programa del listado 3.6 (Sec. 3.1.2.2) que describe un buffer triestado, elabore un programa de un circuito triestado octal como el de las figuras 3.13a) y 3.13b).



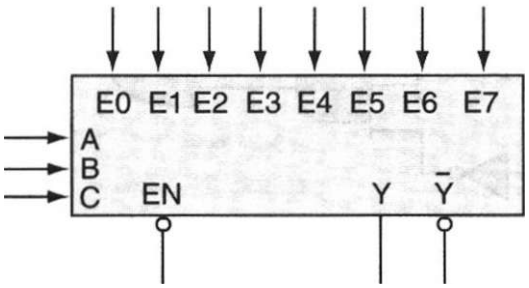
a)



b)

Multiplexores

3.14 Diseñe un programa de un multiplexor de 1 bit con ocho entradas como el que se ilustra en la figura siguiente. Implemente el algoritmo con base en la tabla de verdad adjunta.



a) Multiplexor

EN	c	B	A	Y	Y'
1	X	X	X	0	1
0	0	0	0	E0	E0'
0	0	0	1	El	E1'
0	0	0	0	E2	E2'
0	0	0	1	E3	E3'
0	1	1	0	E4	E4'
0	1	1	1	E5	E5'
0	1	1	0	E6	E6'
0	1	1	1	E7	*E7'*

b) Tabla de verdad

3.15 Diseñe un programa en VHDL que produzca un decodificador binario de 2 x 4 como un circuito demultiplexor de cuatro salidas y un bit. Considere las siguientes especificaciones en el diseño:

a) El circuito debe tener dos señales para seleccionar una salida.
b) El circuito sólo tendrá una señal de entrada.
c) Implemente una señal ENABLE para la habilitación del circuito.

Bibliografía

Floyd T. L.: *Fundamentos de Sistemas Digitales.* Prentice Hall, 1998.

Teres Ll., Torroja Y., Olcoz S. y Villar E.: *VHDL Lenguaje Estándar de Diseño Electrónico.* McGraw-Hill, 1998.

Maxinez G. David, Alcalá Jessica: *Diseño de Sistemas Embebidos a través del Lenguaje de Descripción en Hardware VHDL.* XIX Congreso Internacional Académico de Ingeniería Electrónica. México, 1997.

Kloos C., Cerny E.: *Hardware Description Language and their aplications. Specification, modelling, verification and synthesis of microelectronic systems.* Chapman & Hall, 1997.

IEEE: *The IEEE standard VHDL Language Reference Manual.* IEEE-Std-1076-1987,1988.

Zainalabedin Navabi: *Analysis and Modeling of Digital Systems.* McGraw-Hill, 1988.

Ashenden P J.: *The Designer's guide to VHDL.* Morgan Kauffman Publishers, Inc., 1995.

Lipsett R., Schaefer C.: *VHDL Hardware Description and Design.* Kluwer Academic Publishers, 1989.

Mazor S., Langstraat P: A *guide to VHDL.* Kluwer Academic Publishers, 1993.

Armstrong J.R. y Gail Gray F.: *Structured Design with VHDL.* Prentice Hall, 1997.

Skahill K.: *VHDL for Programmable Logic.* Addison Wesley, 1996.

Bhasker J. A.: A *VHDL Primer.* Prentice Hall, 1992.

Randolph H.: *Applications of VHDL to Circuit Design.* Kluwer Academic Publisher.

Capítulo **4**

Diseño lógico secuencial con VHDL

Introducción

Los circuitos digitales que hemos manejado con anterioridad han sido de tipo combinacional; es decir, son circuitos que dependen por completo de los valores que se encuentran en sus entradas, en determinado tiempo. Si bien un sistema secuencial puede tener también uno o más elementos combinacionales, la mayoría de los sistemas que se encuentran en la práctica incluyen elementos de memoria, los cuales requieren que el sistema se describa en términos de *lógica secuencial.*

En este capítulo se describen algunos de los circuitos secuenciales más utilizados en la práctica, como flip-flops, contadores, registros, etc., además se desarrollan ejercicios para aprender la programación de circuitos secuenciales en los que se integran los conceptos adquiridos en los capítulos anteriores.

4.1 Diseño lógico secuencial

Un sistema secuencial está formado por un circuito combinacional y un elemento de memoria encargado de almacenar de forma temporal la historia del sistema.

En esencia, la salida de un sistema secuencial no sólo depende del valor presente de las entradas, sino también de la historia del sistema, según se observa en la figura 4.1.

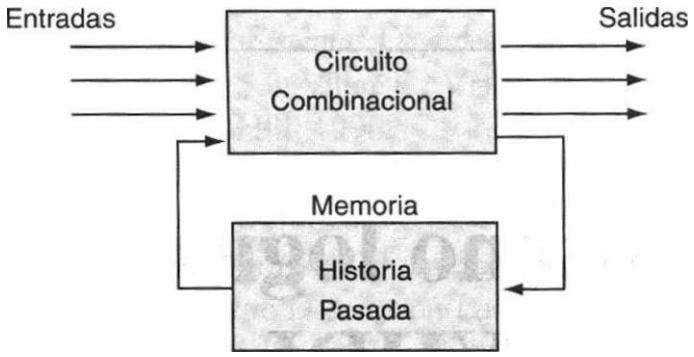


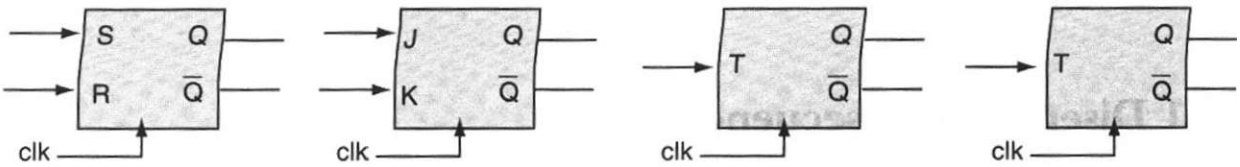
Figura 4.1 Estructura de un sistema secuencial.

Básicamente hay dos tipos de sistemas secuenciales: síncronos y asincronos; el comportamiento de los primeros se encuentra sincronizado mediante el pulso de reloj del sistema, mientras que el funcionamiento de los sistemas asincronos depende del orden y momento en el cual se aplican sus señales de entrada, por lo que no requieren un pulso de reloj para sincronizar sus acciones.

4.2 Flip-flops

El elemento de memoria utilizado indistintamente en el diseño de los sistemas síncronos o asincronos se conoce como flip-flop o celda binaria.

La característica principal de un flip-flop es mantener o almacenar un bit de manera indefinida hasta que a través de un pulso o una señal cambie de estado. Los flip-flops más conocidos son los tipos SR, JK, T y D. En la figura 4.2 se presenta cada uno de estos elementos y la tabla de verdad que describe su comportamiento.



S	R	Q	Qt+1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	·
1	1	1	·

a)

J	K	Q	Qt+1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

b)

D	Q	Qt+1
0	0	0
0	1	0
1	0	1
1	1	1

c)

T	Q	Qt+1
0	0	0
0	1	1
1	0	1
1	1	0

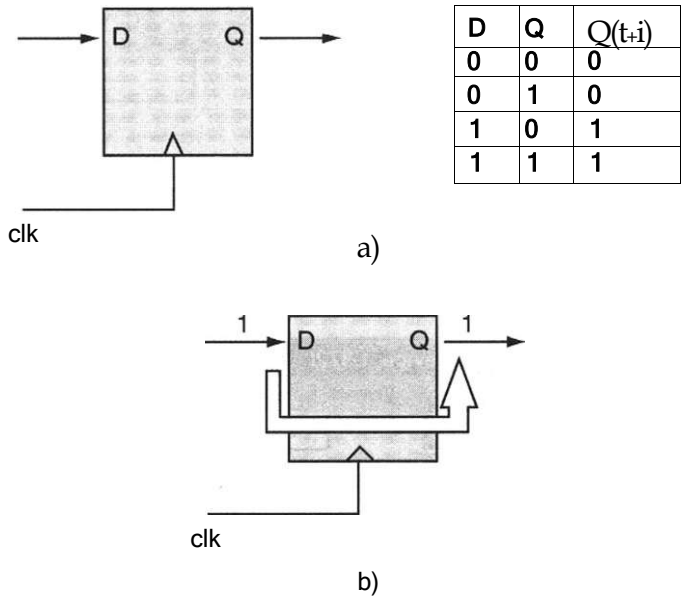
d)

Figura 4.2 Flip-flops y tablas de verdad características.

Es importante recordar el significado de la notación Q y $Q(_t+i)$;

Q = estado presente o actual
Q_t+1 = estado futuro o siguiente

Por ejemplo, consideremos la tabla de verdad que describe el funcionamiento del flip-flop tipo D, mostrado en la figura 4.2c) y que se muestra de nuevo en la figura 4.3a).



D	Q	Q(t+i)
0	0	0
0	1	0
1	0	1
1	1	1

Figura 4.3 a) Diagrama y tabla de verdad del flip-flop D, b) Flujo de información en el dispositivo.

Cuando el valor de la entrada D es igual a 1, figura 4.3b), la salida Q_{t+i} adopta el valor de 1: $Q_t+i = 1$ siempre y cuando se genere un pulso de reloj. Es importante resaltar que el valor actual en la entrada D es transferido a la salida Q_{t+}¡ sin importar cuál sea el valor previo que haya tenido la salida Q en el estado presente.

En el diseño secuencial con VHDL las declaraciones If-then*else son las más utilizadas; por ejemplo, el programa del listado 4.1 usa estas declaraciones.

La ejecución del proceso es sensible a los cambios en *clk* (pulso de reloj); esto es, cuando *clk* cambia de valor de una transición de 0 a 1 *(clk* — 1), el valor de D se asigna a Q y se conserva hasta que se genera un nuevo pulso. A la inversa, si *clk* no presenta dicha transición, el valor de Q se mantiene igual. Esto puede observarse con claridad en la simulación del circuito, fig. 4.4.

```
library ieee;
use ieee.std_logic_1164.all ;
entity f fd is port (
 D,clk: in std_logic;
 Q:     out std_logic) ;
end ffd;
architecture arq_ffd of ffd is
begin
   process (clk) begin
          if (clk'event and clk='l') then
              Q <= D;
          end if ;
   end process ;
end arq_ffd;
```

Listado 4.1 Descripción de un flip-flop disparado por flanco positivo.

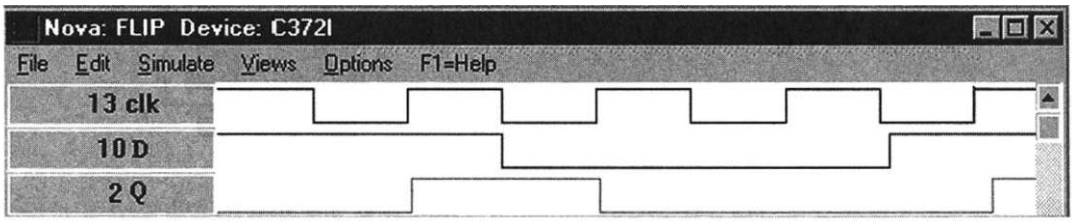


Figura 4.4 Simulación del flip-flop D.

Notemos que la salida Q toma el valor de la entrada D sólo cuando la transición del pulso de reloj es de 0 a 1 y se mantiene hasta que se ejecuta de nuevo el cambio de valor de la entrada *clk.*

Atributo event

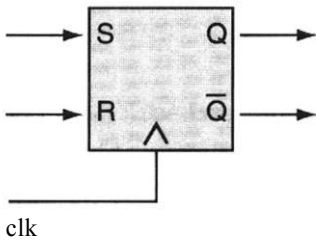
En el lenguaje VHDL los atributos sirven para definir características que se pueden asociar con cualquier tipo de datos, objeto o entidades. El atributo event[1] (evento) se utiliza para describir un hecho u ocurrencia de una señal en particular.

En el código del listado 4-1 podemos observar que la condición if *clk*'event es cierta sólo cuando ocurre un cambio de valor; es decir, un suceso (event) de la señal *clk.* Como se puede apreciar, la declaración (if-then) no maneja la condición else, debido a que el compilador mantiene el valor de Q hasta que no exista un cambio de valor en la señal *clk.*

[1] **El** apóstrofo' indica que se trata de un atributo.

Para mayor información de los atributos predefinidos en VHDL consulte el apéndice C.

Ejemplo 4.1 Escriba un programa que describa el funcionamiento de un flip-flop SR con base en la siguiente tabla de verdad.



S	R	Q	Qt+i
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

Figura E4.1 Tabla de funcionamiento.

Solución

La tabla de verdad del flip-flop SR muestra que cuando la entrada S es igual a 1 y la entrada R es igual a 0, la salida Q_t+i toma valores lógicos de 1. Por otro lado, cuando S = 0 y R=l, la salida Q_t+i = 0; en el caso de que S y R sean ambas igual a 1 lógico, la salida Q_t+j queda indeterminada; es decir, no es posible precisar su valor y éste puede adoptar el 0 o 1 lógico.

Por último, cuando no existe cambio en las entradas S y R —es decir, son igual a 0—, el valor de Q_t+i mantiene su estado actual Q.

Con base en el análisis anterior, el programa en VHDL puede realizarse utilizando instrucciones condicionales y un nuevo tipo de datos: *valores no importa ('-'),* los cuales permiten adoptar un valor de 0 o 1 lógico de manera indistinta (Listado 4-2).

```
1 library ieee;
2 use ieee.std_logic_1164.all ;
3 entity ffsr is port (
4       S,R,clk: in std_logic;
5       Q, Qn: inout std_logic);
6 end ffsr;
7 architecture a_ffsr of ffsr is
8 begin
9 process (elk, S, R)
11 begin
10     if (elk'event and elk = '1') then
11      if (S = ' 0'and R = '1') then
12          Q <= '0';
13            Qn <= '1' ;
14         elsif (S = '1' and R = '0') then
15              Q <= '1';
16             Qn <= '0';
17         elsif (S - '0' and R = '0') then
18              Q <= Q;
19             Qn <= Qn;
20         else
21              Q <=    ;
22             Qn <= '-';
23        end if;
24      end if;
25    end process;
26 end a_ffsr;
```

Listado 4.2 Código VHDL de un registro de 8 bits.

4.3 Registros

En la figura 4.5 se presenta la estructura de un registro de 8 bits con entrada y salida de datos en paralelo. El diseño es muy similar al flip-flop anterior, la diferencia radica en la utilización de vectores de bits en lugar de un solo bit, como se observa en el listado 4.2.

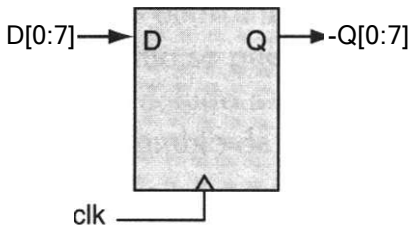


Figura 4.5 Registro paralelo de 8 bits.

```
library ieee;
use ieee.std_logic_1164.all;
entity reg is port (
 D:   in std_logic_vector(0 to 7);
 clk: in std_logic;
 Q:   out std_logic_vector(0 to 7));
end reg;
architecture arqreg of reg is
begin
   process (clk) begin
   if (clk'event and clk='1') then
       Q <= D;
   end if;
 end process;
end arqreg;
```

Listado 4.2 Código VHDL de un registro de 8 bits.

Ejemplo 4.2 Escriba un programa de un registro de 4 bits, como el que se muestra en la figura E4.2. Realice el diseño utilizando instrucciones if-then-else y procesos.

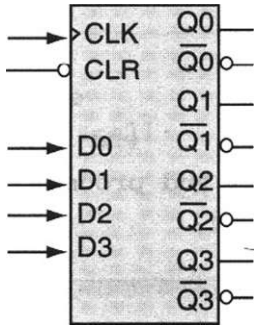


Figura E4.2

Solución

Como puede observarse en la tabla que describe el comportamiento del circuito, si CLR = 0, las salidas Q adoptan el valor de 0; pero si CLR — 1, toman el valor de las entradas DO, DI, D2 y D3.

CLR	D	Q	QN
0	*	0	1
1	Dn	Dn	Dn

El código del programa se observa en el siguiente listado.

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity reg4 is port(
4            D: in std_logic_vector(3 downto 0);
5  CLK,CLR: in std_logic;
6     Q/Qn: inout std_logic_vector(3 downto 0));
7 end reg4;
8 architecture a_reg4 of reg4 is
9 begin
10          process (CLK,CLR) begin
11              if (CLK'event and CLK ='1') then
12                  if (CLR = '1' )then
13                      Q <= D;
14                     Qn <= not Q;
15                  else
16                      Q <= "0000";
17                     Qn <= "1111";
18                   end if;
19               end if;
20          end process;
21 end a_reg4;
```

Listado E4.2

Las variables sensitivas que determinan el comportamiento del circuito se encuentran dentro del proceso (CLR y CLK). Para transferir los datos de entrada hacia la salida es necesario, según se expuso, generar un pulso de reloj a través del atributo event (CLK'revent and CLK = T)

4.4 Contadores

Los contadores son entidades muy utilizadas en el diseño lógico. La forma usual para describirlos en VHDL es mediante operaciones de incremento, decremento de datos o ambas.

Como ejemplo veamos la figura 4.6 que representa un contador ascendente de 4 bits, así como el diagrama de tiempos que muestra su funcionamiento.

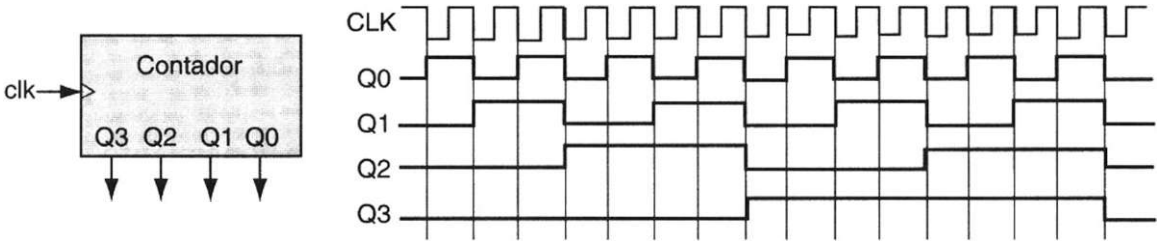


Figura 4.6 Contador binario de cuatro bits.

Cabe mencionar que la presentación del diagrama de tiempos de este circuito tiene la finalidad de ilustrar el procedimiento que se sigue en la programación, ya que puede observarse con claridad el incremento que presentan las salidas cuando se aplica un pulso de reloj a la entrada (listado 4.3).

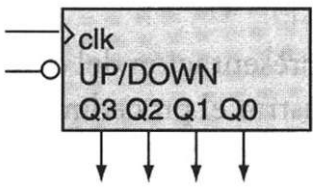
```
library ieee;
use ieee.std_logic_1164.all ;
use work.std_ari th.all ;
entity cont4 is port (
 elk: in std_logic;
   Q: inout std_logic_vector(3 downto 0));
end cont4;
architecture arqeont of cont4 is
begin
    process (elk)
       begin
          If (elk'event and elk = '1') then
              Q <= Q + 1;
          end if;
      end process ;
end arqeont;
```

Listado 4.3 Código que describe un contador de 4 bits.

Cuando requerimos la retroalimentación de una señal (Q <= Q+1), ya sea dentro o fuera de la entidad, utilizamos el modo inout (Vea sección de Modos en Capítulo 2). En nuestro caso el puerto correspondiente a Q se maneja como tal, debido a que la señal se retroalimenta en cada pulso de reloj. Notemos también el uso del paquete std_arith que, como ya se mencionó, permite usar el operador + con el tipo std_logic_vector.

El funcionamiento del contador se define básicamente en un proceso, en el cual se llevan a cabo los eventos que determinan el comportamiento del circuito. Al igual que en los otros programas, una transición de 0 a 1 efectuada por el pulso de reloj provoca que se ejecute el proceso, lo cual incrementa en 1 el valor asignado a la variable Q. Cuando esta salida tiene el valor de 15 ("1111") y si el pulso de reloj se sigue aplicando, el programa empieza a contar nuevamente de 0.

Ejemplo 4.3 Elabore un programa que describa el funcionamiento de un contador de 4 bits. Realice en el diseño una señal de control (Up/Down) que determine el sentido del conteo: ascendente o descendente (Fig. E4.3).



UP/DOWN	ACCION
0	Cuenta ascendente
1	Cuenta descendente

Figura E4.3 Contador binario de cuatro bits.

Solución

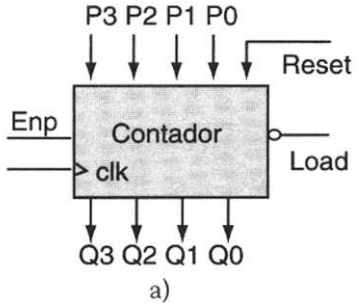
La señal de control Up/Down permite definir si el conteo se realiza en sentido ascendente o descendente. En nuestro caso, un cero aplicado a esta señal determina una cuenta ascendente: del 0 al 15. De esta forma, el funcionamiento del circuito queda determinado por dos señales: el pulso de reloj (clk) y la señal Up/Down, como se ve en el siguiente programa.

```
library ieee;
use ieee.std_logic_1164.all ;
use work.std_arith.all;
entity contador is port (
 clk: in std_logic;
  UP:in std_logic;
   Q: inout std_logic_vector(3 downto 0));
end contador;
architecture a_contador of contador is
begin
 process (UP, clk) begin
    if (clk'event and clk = '1')then
        if (UP = '0') then
              Q <= Q+1;
        else
              Q <= Q-1;
        end if ;
    end if ;
 end process ;
end a_contador;
```

Listado E4.3.

Contador con reset y carga en paralelo (load)

La entidad de diseño que aparece en la figura 4.7a) es un ejemplo de un circuito contador síncrono de 4 bits. Este contador tiene varias características adicionales respecto al anterior. Su funcionamiento se encuentra predeterminado por los valores que se hallan en la tabla de la figura 4.7b). Como se puede observar, este contador posee las entradas de control Enp y Load y según el valor lógico que tengan en sus terminales realizarán cualesquiera de las operaciones reseñadas en la tabla. Note que la señal de Reset se activa en nivel alto de forma asincrona; por último, el circuito tiene cuatro entradas en paralelo declaradas como P3, P2, PI y P0.



a)

Enp	Load	Accion
0	0	Carga
0	1	Mantiene Estado
1	0	Carga
1	1	Cuenta

b)

Figura 4.7 a) Contador binario de cuatro bits, **b)** Tabla de funcionamiento.

El propósito de describir este ejemplo radica en subrayar el uso de varias condiciones, de tal manera que no sea posible evaluar una instrucción si la(s) condición(es) predeterminada(s) no se cumple(n). Por ejemplo, observemos el listado 4.4, donde la línea 11 presenta la lista sensitiva de las variables que intervienen en el proceso (clk, ENP, LOAD, RESET). Nótese que no importa el orden en que se declaran. En la línea 12 se indica que si RESET = 1, las salidas Q toman el valor de 0; si no es igual a 1, se pueden evaluar las siguientes condiciones (es importante resaltar que la primera condición debe ser RESET). En la línea 15 se observa el proceso de habilitar una carga en paralelo, por lo que si LOAD — 0 y ENP es una condición de no importa ('-') que puede tomar el valor de 0 o 1 (según la tabla 4.7b), las salidas Q adoptarán el valor que se halle en las entradas P (P3, P2, Pl, P0). Este ejemplo muestra de forma didáctica el uso de la declaración elsif (si no-si).

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use work.std_arith.all;
4 entity cont is port(
5       P: in std_logic_vector(3 downto 0) ;
6       Clk,LOAD,ENP,RESET: in std_logic;
7       Q: inout std_logic_vector(3 downto 0));
8 end cont;
9  architecture arq_cont of cont is
10 begin
11   process (elk, RESET,LOAD, ENP) begin
12     if (RESET = '1') then
13        Q <= "0000";
14     elsif (elk'event and elk = '1') then
15               if  (LOAD = '0' and ENP = '-') then
16                  Q <= P;
17               elsif (LOAD = '!' and ENP = '0') then
18                    Q <= Q;
19               elsif (LOAD = '1' and ENP = '1') then
20                      Q <= Q + 1;
21               end if;
22     end if;
23   end process;
24 end arq_cont;
```

Listado 4.4 Contador con reset, enable y carga en paralelo.

4.5 Diseño de sistemas secuenciales síncronos

Como ya se mencionó, la estructura de los sistemas secuenciales síncronos basa su funcionamiento en los elementos de memoria conocidos como flip-flops. La palabra sincronía se refiere a que cada uno de estos elementos de memoria que interactúan en un sistema se encuentran conectados a la misma señal de reloj, de forma tal que sólo se producirá un cambio de estado en el sistema cuando ocurra un flanco de disparo o un pulso en la señal de reloj.

Existe una división en el diseño de los sistemas secuenciales que se refiere al momento en que se producirá la salida del sistema.

- En la estructura de Mealy (Fig. 4.8) las señales de salida dependen tanto del estado en que se encuentra el sistema, como de la entrada que se aplica en determinado momento.
- En la estructura de Moore (Fig. 4.9) la señal de salida sólo depende del estado en que se encuentra.

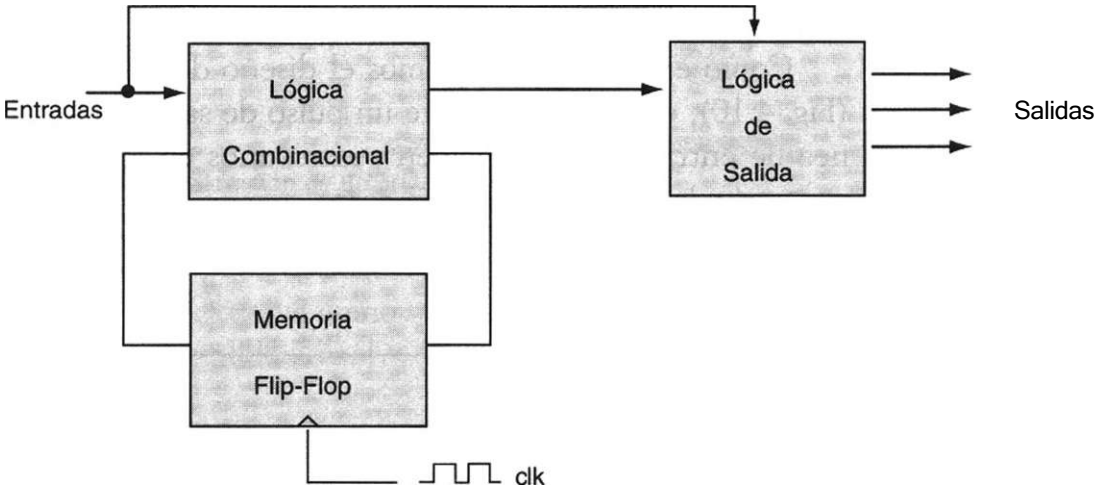


Figura 4.8 Arquitectura secuencial de Mealy.

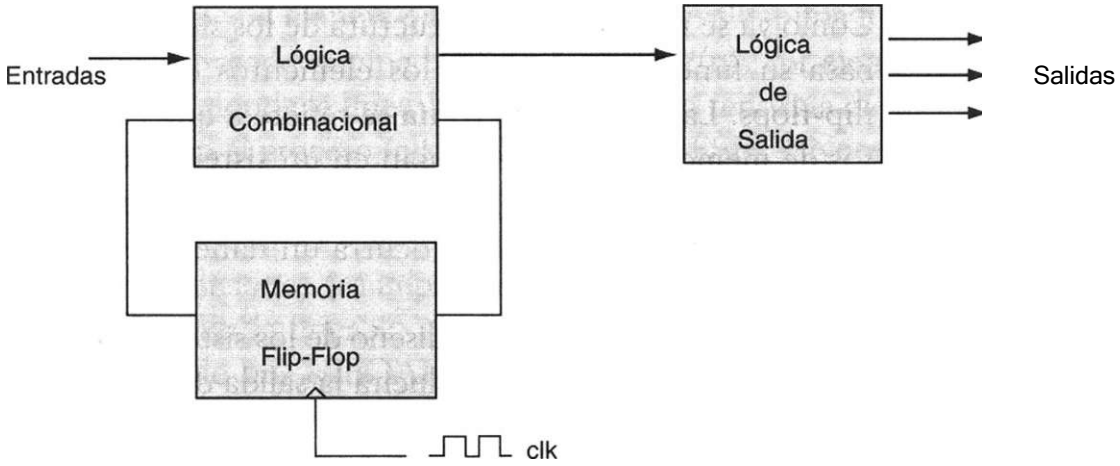


Figura 4.9 Arquitectura secuencial de Moore.

Un sistema secuencial se desarrolla a través de una serie de pasos generalizados que comprenden el enunciado del problema, diagrama de estados, tabla de estados, asignación de estados, ecuaciones de entrada a los elementos de memoria y diagrama electrónico del circuito.

Como ejemplo consideremos el diseño del siguiente sistema secuencial (Fig, 4.10), en el cual se emite un pulso de salida Z (Z=1) cuando en la línea de entrada (X) se reciben cuatro unos en forma consecutiva; en caso contrario, la salida Z es igual a cero.

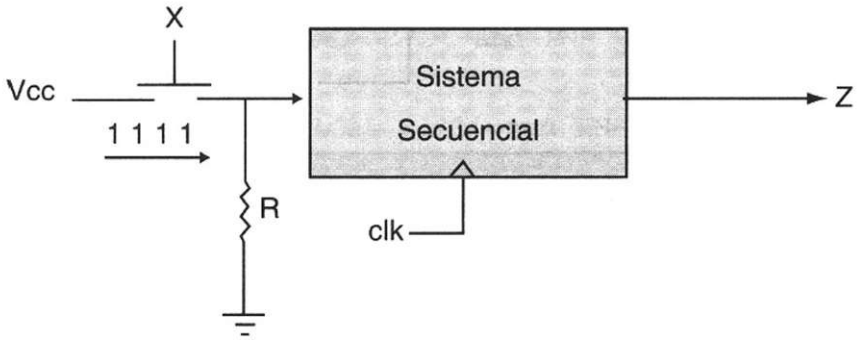


Figura 4.10 Detector de secuencia.

Diagramas de estado

El uso de diagramas de estados en la lógica programable facilita de manera significativa la descripción de un diseño secuencial, ya que no es necesario seguir la metodología tradicional de diseño. En VHDL se puede utilizar un modelo funcional en que sólo se indica la transición que siguen los estados y las condiciones que controlarán el proceso.

De acuerdo con nuestro ejemplo (Fig. 4.12), vemos que el sistema secuencial se puede representar por medio del diagrama de estados de la figura 4.12a): arquitectura Mealy. En este diagrama se advierte que el sistema cuenta con una señal de entrada denominada X y una señal de salida Z. En la figura 4.12b) se muestra la tabla de estados que describe el comportamiento del circuito.

Cuando se está en el estado dO y la señal de entrada X es igual a uno, se avanza al estado di y la salida Z durante esta transición es igual a cero; en caso contrario, cuando la entrada X es igual a cero, el circuito se mantiene en el estado dO y la salida también es cero (Fig. 4.11).

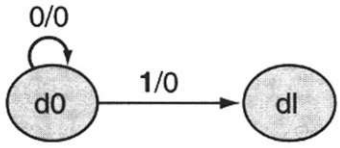
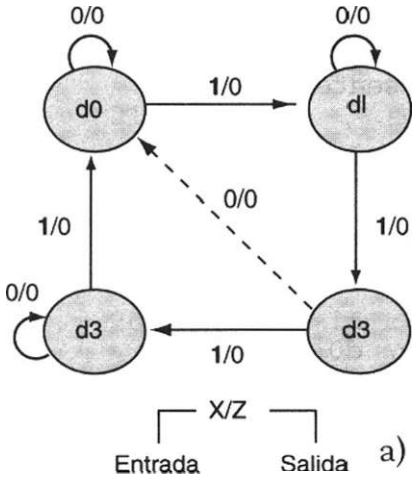


Figura 4.11 Transición de un estado a otro.

Con un poco de lógica se puede intuir el comportamiento del diagrama de estados. Observe como sólo la secuencia de cuatro unos consecutivos provoca que la salida Z = 1.



Edo. presente	Edo. futuro X=0	Edo. futuro X=1	Salida Z X=0	Salida Z X=1
dO	dO	d1	0	0
d1	d1	d2	0	0
d2	d2	d3	0	0
d3	d3	dO	0	1

b)

Figura 4.12 a) Diagrama de Estados, b) Tabla de estados.

Este diagrama se puede codificar con facilidad mediante una descripción de alto nivel en VHDL. Esta descripción supone el uso de declaraciones case-when las cuales determinan, en un caso particular, el valor que tomará el siguiente estado. Por otro lado, la transición entre estados se realiza por medio de declaraciones if-then'else, de tal forma que éstas se encargan de establecer la lógica que seguirá el programa para realizar la asignación del estado.

Como primer paso en nuestro diseño, consideremos los estados dO, di, d2 y d3. Para poder representarlos en código VHDL, hay que definirlos dentro de un tipo de datos enumerado[2] (apéndice C) mediante la declaración type. Observemos la forma en que se listan los identificadores de los estados, así como las señales utilizadas para el estado actual (edo_presente) y siguiente (edofuturo):

```
type estados is (dO, di, d2,d3) ;
signal edo_presente, edo_futuro : estados;
```

El siguiente paso consiste en la declaración del proceso que definirá el comportamiento del sistema. En éste debe considerarse que el edo_futuro depende del valor del edo_presente y de la entrada X. De esta manera la lista sensitiva del proceso quedaría de la siguiente forma:

```
procesol: process (edo_presente, X)
```

Dentro del proceso se describe la transición del edo_presente al edo_futuro. Primero se inicia con la declaración case que especifica el primer estado que se va a evaluar —en nuestro caso consideremos que el análisis comienza en el estado dO (when dO), donde la salida Z siempre es cero sin importar el valor de X. Si la entrada X es igual a 1 el estado futuro es di; en caso contrario, es dO.

De este modo, la declaración del proceso quedaría de la siguiente manera:

```
procesol: process (edo_presente, X) begin
      case edo__presente is
         when dO =>  Z<= '0';
           if X   1' then
              edo_futuro <= di;
           else
              edo_futuro <= dO;
           end if;
```

2 Se llaman tipos enumerados porque en ellos se agrupan o enumeran elementos que pertenecen a un mismo género.

Nótese que en cada estado debe indicarse el valor de la salida (Z < = 0) después de la condición when, siempre y cuando la variable Z no cambie de valor.

En el listado 4.5 se muestra la definición completa del código explicado. Como podemos observar, en el programa se utilizan dos procesos. En el primero, proceso1 (línea 11) se describe la transición que sufren los estados y las condiciones necesarias que determinan dicha transición. En el segundo, proceso2 (línea 42) se lleva a cabo de manera síncrona la asignación del estado futuro al estado presente, de suerte que cuando se aplica un pulso de reloj, el proceso se ejecuta.

En la línea 31 se describe la forma de programar la salida Z en el estado d3 cuando ésta obtiene el valor de 0 o 1, según el valor de la entrada X.

```
1  library ieee;
2  use ieee.std_logic_1164 . all;
3  entity diagrama is port(
4   clk,x:  in std_logic;
5        z:  out std_logic) ;
6  end diagrama;
7  architecture arq_diagrama of diagrama is
8    type estados is (dO, di, d2, d3);
9    signal edo_presente, edo_futuro: estados;
10   begin
11    procesol: process (edo_presente, x) begin
12     case edo__presente is
13        when dO =>  z <= '0';
14          if x =' 1' then
15             edo_futuro <= di;
16          else
17             edo_futuro <= dO;
18          end if;
19        when di =>  z <='0';
20          if x='l' then
21             edo_futuro <= d2;
22          else
23             edo_futuro <= di;
24          end if;
25        when d2 =>  z <='0';
26          if x='l' then
27             edo_futuro <= d3;
28          else
29             edo_futuro <= dO;
30          end if;
```

Continúa

```
31          when d3 =>
32            if x='1' then
33                edo_futuro <= d0;
34                  z <='1';
35          else
36                edo_futuro <= d3;
37                  z <= '0';
38           end if;
39       end case;
40  end process procesol;
41
42   proceso2: process(clk) begin
43     if  (clk'event and clk='1')  then
44          edo_presente <= edo_futuro;
45     end if;
46    end process proceso2;
47  end arq_diagrama ;
```

Listado 4.5 Diseño de un diagrama de estados.

Ejemplo 4.4 Se requiere programar el diagrama de estados de la figura E4-4.

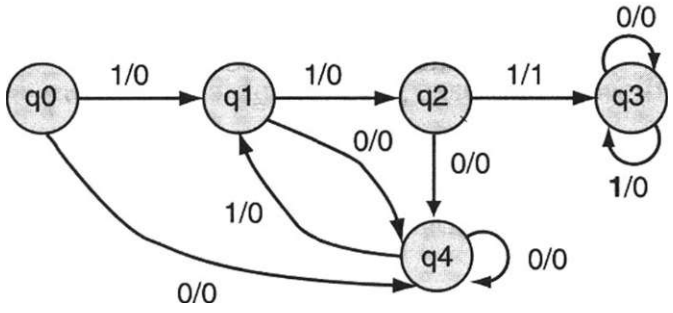


Figura E4.4 Diagrama de estados.

Solución

Por ejemplo, en la figura E4-4 notamos que el circuito pasa al estado ql con el primer 1 de entrada y al estado q2 con el segundo. Las salidas asociadas con estas dos entradas son 0, según se señala. La tercera entrada consecutiva de 1 genera un 1 en la salida y hace que el circuito pase al estado q3. Una vez que se encuentra en q3, el circuito permanecerá en este estado, emitiendo salidas 0.

En lo que respecta a la programación en VHDL, este ejemplo sigue la misma metodología que el ejercicio anterior (Fig. 4.12), la cual consiste en usar estructuras case - when y tipo de datos enumerado, que en este caso contiene los cinco estados (qO, q1, q2, q3 y q4) que componen el diagrama.

El listado correspondiente al programa se encuentra a continuación listado E4.4.

```
library ieee;
use  ieee.std_logic_1164.all;
entity diag is port (
 clk,x: in std_logic;
 z: out std_logic);
end diag;
architecture arq_diag of diag is
 type estados is  (q0,ql/q2,q3/q4);
 signal edo_pres, edo_fut: estados;
  begin
      procesol: process  (edo_pres,x) begin
        case edo_pres is
           when qO => z <= '0';
             if x = '0' then
               edo_fut <= q4;
             else
               edo_fut <= ql;
             end if;
              when ql => z <= '0';
              if x = '0' then
                edo_fut <= q4;
              else
                edo_fut <= q2;
              end if;
           when q2 =>
              if x = '0' then
                edo_fut <= q4;
                z <= '0';
              else
                edo_fut <= q3;
                z <= '1';
              end if;
           when q3 => z <= '0';
              if x = '0' then
```

Continúa

```
                edo_fut <= q3;
            else
                edo_fut <= q3;
            end if;
        when q4 => z <= '0';
            if x = '0' then
                edo_fut <= q4;
            else
                edo_fut <= ql;
            end if;
    end case;
end process procesol;
proceso2: process (clk) begin
        if (clk'event and clk='l')then
            edo_pres <= edo_fut;
        end if;
    end process proceso2;
end arqL_diag;
```

Listado E4.4.

Ejercicios

Flip-Flop

4.1 Realice un programa en VHDL que describa el funcionamiento del flip-flop tipo JK. Auxíliese para su descripción con la tabla característica del Flip-Flop.

J	K	Q	Qt+i
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Tabla característica del Flip-Flop JK

4.2. Realice un programa en VHDL que describa el funcionamiento del flip-flop tipo T. Auxíliese para su descripción con la tabla característica del Flip-Flop.

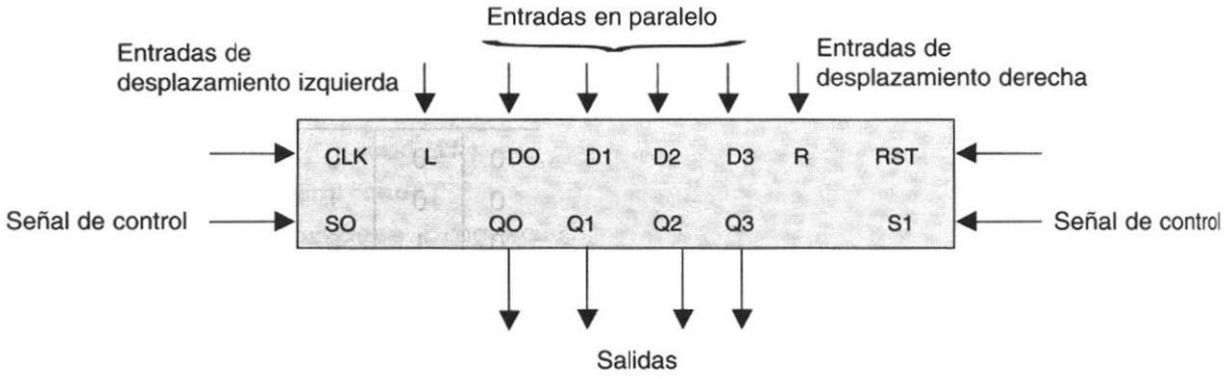
T	Q	Qt+i
0	0	0
0	1	1
1	0	1
1	1	0

Tabla característica del Flip-Flop T

Registros

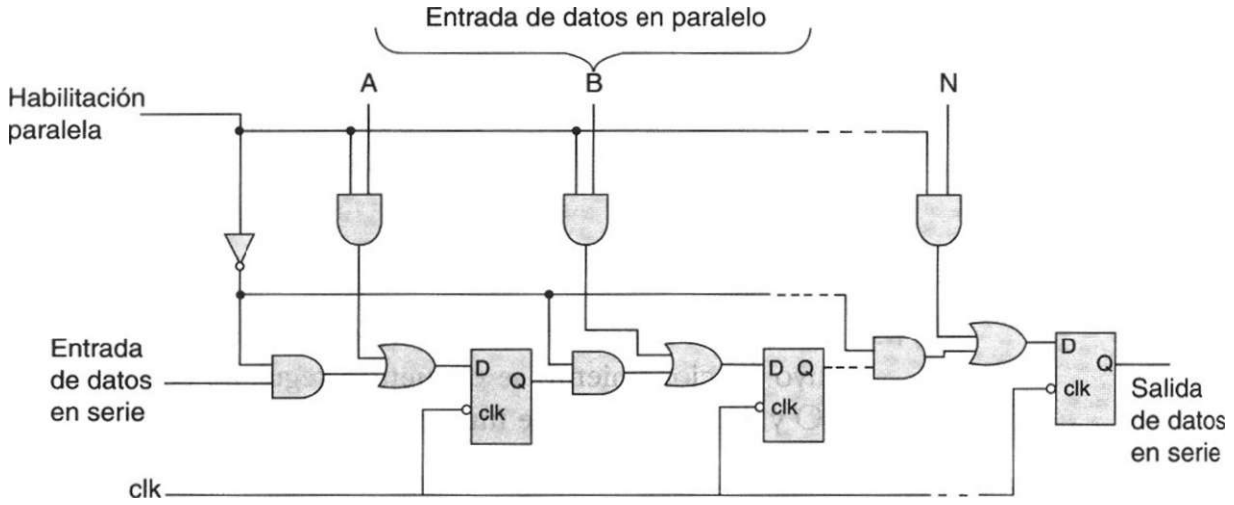
4.3. Diseñe un registro de 4 bits como el mostrado en la siguiente figura y cuyo funcionamiento se encuentra regulado por las señales de control SO y SI, tal y como se muestra en la siguiente tabla:

SO	SI	ACCION
0	0	Hold (Reten)
0	1	Desplazamiento Izquierda SL
1	0	Load (Carga)
1	1	Desplazamiento Derecha SR

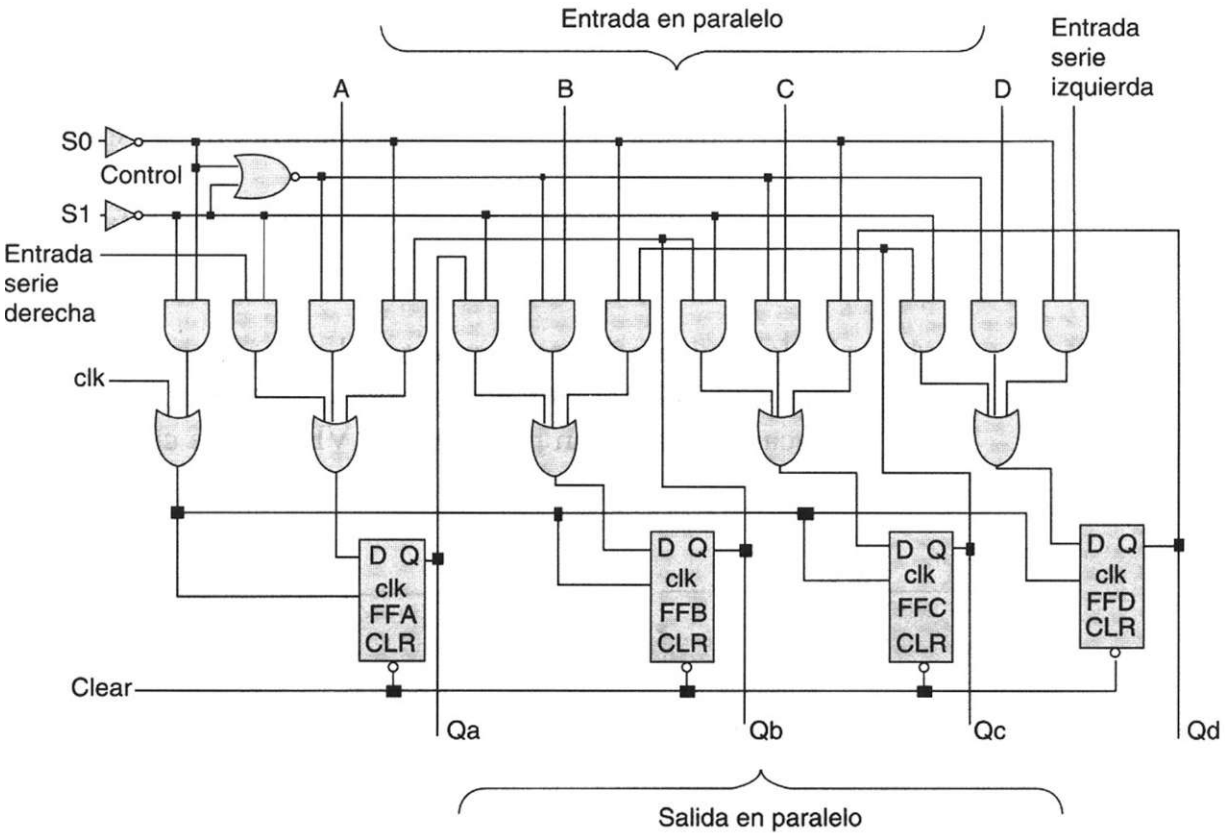


CLK= Reloj
SO, S1 = Señales de control
DO...03= Entradas de datos
RST= Reset,
Q0...Q3 = Salida de datos
L= Entrada serie desplazamiento a la izquierda
R= Entrada serie desplazamiento a la derecha.

4-4 En la figura siguiente se muestra el esquema lógico de un registro de desplazamiento con entrada serie/ paralelo y salida serie. Realice un programa en VHDL que realice la misma función.

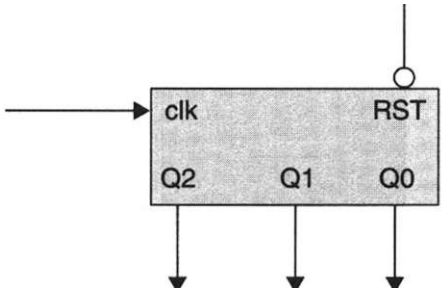


4.5 En la figura siguiente se muestra el esquema lógico de un registro de desplazamiento con salida en paralelo. La tabla de funcionamiento correspondiente se muestra a continuación. Realice un programa en VHDL que realice la función del circuito.



Contadores

4.6 Diseñe y programe un contador que realice la secuencia 0,1,3, 5 y repita el ciclo. El circuito debe contar con una señal de reset activo en bajo, que coloca las salidas Q en estado bajo.

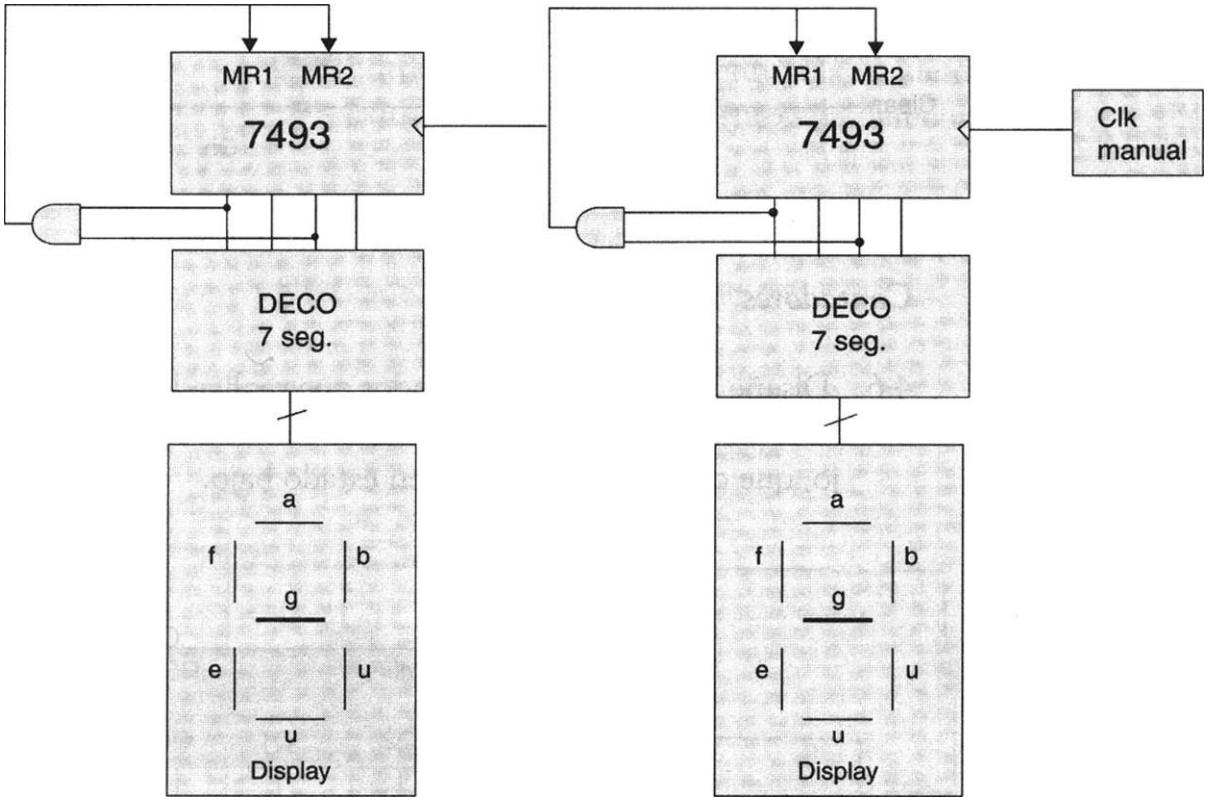


4.7 Diseñe y programe un contador que realice la secuencia 0,1,2,3,4,5,6,7 y repita el ciclo. El circuito debe contar con una señal de reset activo en bajo, que coloca las salidas Q en estado bajo.

4.8 Programe un circuito contador ascendente / descendente del 0 al 3 mediante una señal de control X. Si X=0, el contador cuenta ascendente, si X= 1, el contador cuenta descendente.

4.9 Programe un circuito contador ascendente / descendente del 0 al 15 mediante una señal de control X. Si X =0, el contador cuenta ascendente; si X=1, el contador cuenta descendente.
Existen dos señales de salida denominadas Z1 y Z2 que se activan de la siguiente forma:
Z1 = 1 Cuando el contador se encuentra en los estados pares, en caso contrario Z1 = 0.
Z2 = 1 Cuando el contador se encuentra en los estados impares, en caso contrario 72=0

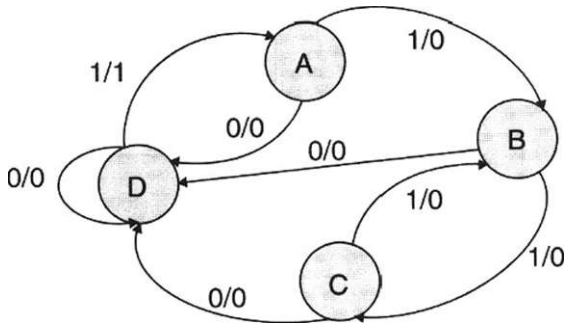
4.10 En la siguiente figura se muestra el cronómetro digital configurado para contar del 0 al 99 mediante dos contadores SN 7493 conectados en cascada. Realice un programa en VHDL que cuente del 0 al 99.



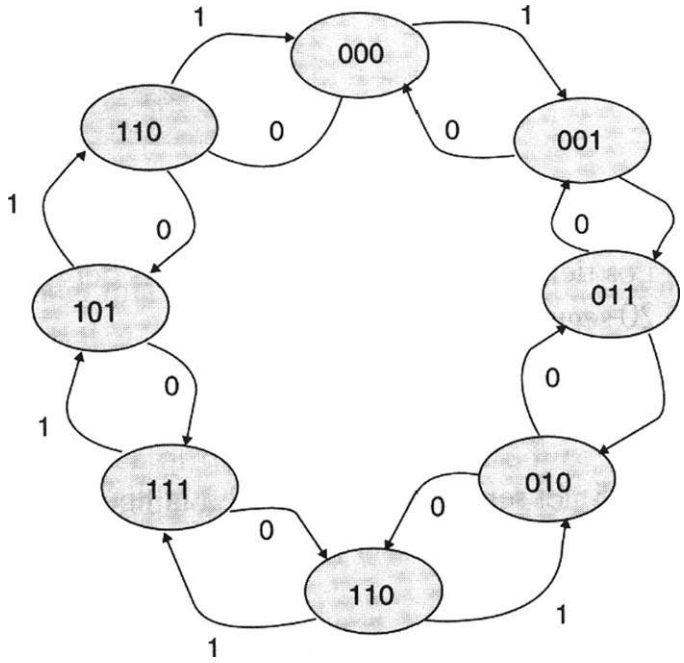
4-11 Realice un programa para un cronómetro que debe contar del 0 al 245 y repita el ciclo.

Sistemas secuenciales

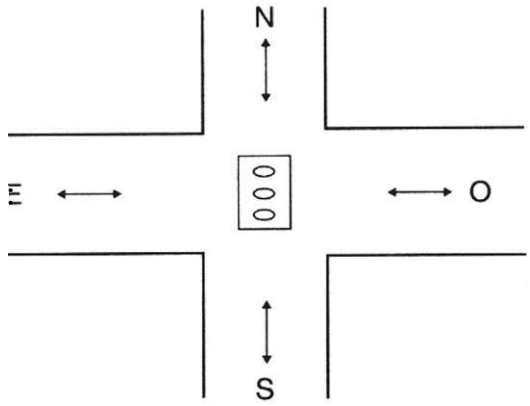
4.12 Realice un programa que resuelva el siguiente diagrama de estados:



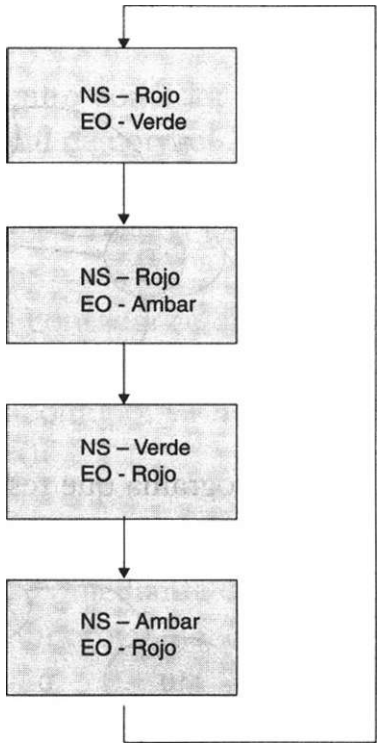
4.13 Realice un programa que resuelva el siguiente diagrama de estados:



4.14 En la figura se muestra el crucero de una avenida controlada a través de un semáforo.



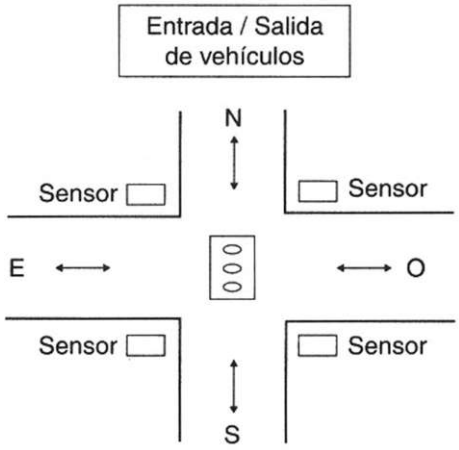
Los automóviles pueden circular en la dirección NS o EO mediante la siguiente secuencia:



Los tiempos de duración de las luces del semáforo son: rojo 20 segundos, verde 20 segundos, ámbar 5 segundos.

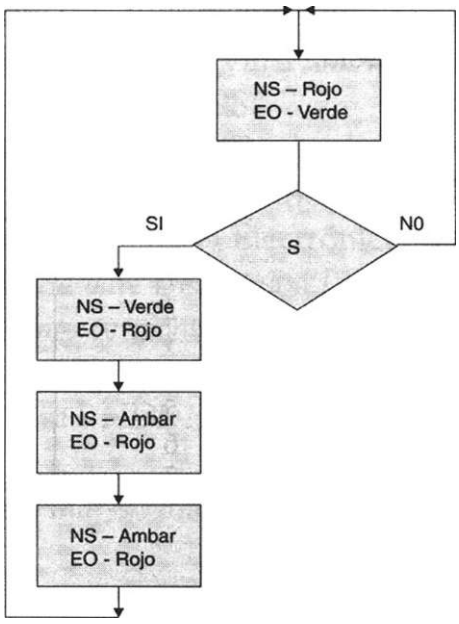
Proponga el diagrama de estados correspondiente y realice el programa del sistema secuencial.

4.15 El semáforo mostrado en la figura del ejercicio de la página 117 ha sido modificado tal y como se muestra en la siguiente figura.



Se requiere agilizar la entrada y salida de vehículos por medio de sensores (S) que detectan la entrada o salida de ellos. Generalmente el semáforo EO siempre se encuentra en verde y en contraparte el semáforo NS está en

rojo; cuando alguno de los sensores detecta la entrada o salida del vehículo envía una señal al sistema que gobierna el semáforo para que éste realice la secuencia siguiente:



Nuevamente la duración de encendido de los focos del semáforo son: roja 20 segundos, verde 20 segundos, ámbar 5 segundos.

Proponga el diagrama de estados correspondiente y realice el programa del sistema secuencial.

4.16 Programe un detector de secuencia que produce una salida Z=1 sólo cuando en la señal de entrada X se produce la siguiente secuencia X = 110011.

4-17 Programe un detector de secuencia cuya salida Z= 11011111 cuando aplicamos a la entrada la secuencia X= 01101010.

4.18 Realice el programa correspondiente al sistema secuencial especificado en la siguiente tabla.

Presente	X = 0	X = 1
	Futuro	
A	**B**/0	E/0
B	A/1	C/1
C	B/0	C/1
D	C/0	E/0
E	D/1	A/0

Considere la siguiente asignación de estados:

A 000 B001 C010 D 011 E 100

4.19 Realice el programa de un contador ascendente /descendente de números seudo aleatorios de 3 bits. El circuito tiene una entrada de control X .Cuando X = 0 , el contador cuenta ascendente , si X = 1 el contador genera números seudo aleatorios, tal y como se muestra en la tabla.

Presente	X = 0	X = 1
	Futuro	
0	1	0
1	2	4
2	3	5
3	4	1
4	5	2
5	6	6
6	7	7
7	0	3

4.20 Programe el sistema secuencial correspondiente a una máquina despachadora de refrescos, el valor del refresco es de $ 5.00 pesos y la máquina acepta monedas de $5.00, $ 10.00 y $ 20.00. Cuando se introduce una moneda de diez o veinte pesos, la máquina debe dar cambio en monedas de $5.00 pesos hasta completar el cambio correspondiente.

Para realizar este programa no considere el sistema detector de monedas ni el sistema de servicio encargado de dar el refresco.

Programe exclusivamente el sistema secuencial encargado de controlar la secuencia descrita anteriormente.

Bibliografía

T.L. Floyd: *Fundamentos de* sistemas *digitales.* Prentice Hall, 1998.

Ll. Teres; Y. Torroja; S. Olcoz; E. Villar: *VHDL Lenguaje estándar de diseño electrónico.* McGraw-Hill, 1998.

David G. Maxinez, Jessica Alcalá: *Diseño de Sistemas Embebidos a través del Lenguaje de Descripción en Hardware VHDL.* XIX Congreso Internacional Académico de Ingeniería Electrónica. México, 1997.

C. Kloos, E. Cerny: *Hardware Description Language and their Aplications. Specification, modelling, verification and synthesis of microelectronic systems.* Chapman&Hall, 1997.

IEEE: *The IEEE standard VHDL Language Reference Manual.* IEEE-Std-1076-1987, 1988.

Zainalabedin Navabi: *Analysis and Modeling of Digital Systems.* McGraw-Hill, 1988.

E J. Ashenden: *The Designer's guide to VHDL.* Morgan Kauffman Publishers, Inc., 1995.

R. Lipsett, C. Schaefer: *VHDL Hardware Description and Design.* Kluwer Academic Publishers, 1989.

S. Mazor, P Langstraat: A *guide to VHDL.* Kluwer Academic Publishers, 1993.

J.R. Armstrong y F. Gail Gray: *Structured Design with VHDL.* Prentice Hall, 1997.

K. Skahill: *VHDL for Programmable Logic.* Addison Wesley, 1996.

J. A. Bhasker: A *VHDL Primer.* Prentice Hall, 1992.

H. Randolph: *Applications of VHDL to Circuit Design.* Kluwer Academic Publisher.

Capítulo 5

Integración de entidades en VHDL

Introducción

Hasta este momento hemos utilizado la programación en VHDL para diseñar entidades individuales (bloques lógicos mínimos), con el único objeto de familiarizarlo con los diversos estilos de diseño y programación o ambos, así como con el uso y aplicación de las palabras reservadas en VHDL.

Sin embargo, es obvio que esta herramienta de diseño no fue creada para este fin. Como vimos en el capítulo 1, existen varias razones para su utilización; pero quizá su verdadera fortaleza radica en que permite integrar "sistemas digitales" que contienen una gran cantidad de subsistemas electrónicos con el fin de minimizar el tamaño de la aplicación. En primera instancia, en un solo circuito integrado y si el problema es muy complejo, a través de una serie sucesiva de circuitos programables, sea que se llamen CPLD (dispositivo lógico programable complejo) o FPGA (arreglos de compuertas programables en campo).

5.1 Esquema básico de integración de entidades

La integración de entidades puede realizarse mediante el diseño individual de cada bloque lógico a través de varios procesos internos que posteriormente pueden unirse mediante un programa común. Otra posibilidad es observar y analizar de manera global todo el sistema evaluando su comportamiento sólo a través de sus entradas y salidas. En ambos casos el resultado es satisfactorio; más bien, nuestra tarea consiste en analizar las ventajas y desventajas que existen en ambas alternativas de solución. En el primer caso, el inconveniente principal es el número excesivo de terminales utilizadas en el dispositivo, debido a que al diseñar entidades individuales, se tendrían que declarar las terminales de entrada-salida de cada entidad (Fig 5.1a).

En el segundo caso, cuando observamos el sistema como un todo (Fig. 5.1b) el número de terminales de entrada y salida disminuye, por lo que nuestro trabajo es desarrollar no sólo un algoritmo interno capaz de interpretar el funcionamiento de cada bloque, sino también cómo conectar cada uno de ellos.

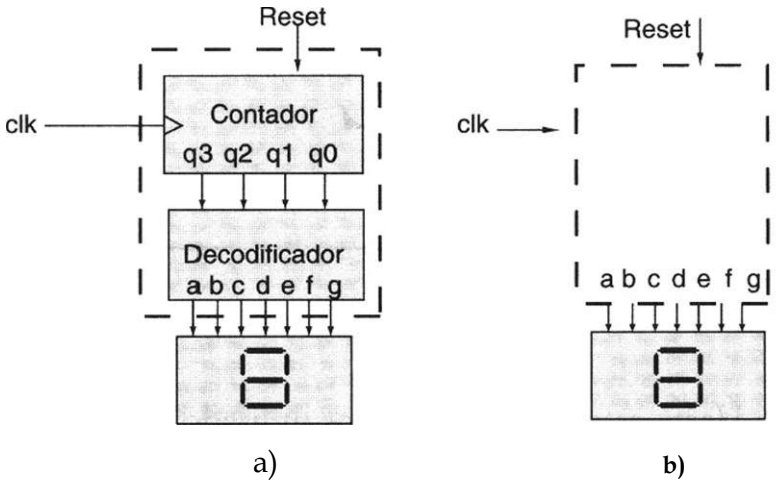


Figura 5.1 a) Diseño mediante entidades individuales, **b)** Diseño mediante la relación de entradas/salidas.

5.1.1 Programación de entidades individuales

Para ejemplificar las posibles diferencias, consideremos la programación del contador y decodificador de la figura 5.1 a), mediante la programación de entidades individuales, listado 5.1.

```
— Programación de entidades individuales
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use work.std_arith.all;
4  entity display is port (
5  clk,reset: in std_logic;
6  d: inout std_logic_vector(6 downto 0);— salidas del decodificador
7  q: inout std_logic_vector(3 downto 0));— salidas del contador
8  end display;
9  architecture arqdisplay of display is
10 begin
11     process (clk,reset)
12       begin
13          If (clk'event and clk = '1') then
14               q <= q + 1;
```

Continúa

```
15                  if (reset = '1' or q = "1001") then
16                    q <= "0000";
17                  end if;
18          end if;
19      end process;
20
21    process (q) begin
22   case q is
23    when "0000" => d <= "1111110";
24    when "0001" => d <= "0110000";
25    when "0010" => d <= "1101101";
26    when "0011" => d <= "1111001";
27    when "0100" => d <= "0110011";
28    when "0101" => d <= "1011011";
29    when "0110" => d <= "1011111";
30    when "0111" => d <= "1110000";
31    when "1000" => d <= "1111111";
32    when "1001" => d <= "1110011";
33    when others => d <= "0000000";
34    end case;
35  end process;
36 end arqdisplay;
```

Listado 5.1 Integración de un contador y un decodificador en el mismo programa.

Como podemos observar, en la declaración de la entidad (*entity*) se han asignado las terminales de entrada y salida correspondientes a cada bloque lógico de manera individual; así por ejemplo, el contador tiene las salidas q[q3, q2, q1, qO] y el decodificador las salidas d(da, db, de, dd, de, df, dg). Note que la programación requiere dos procesos: el primero (líneas 11 a 19) describe el funcionamiento del contador; el segundo (líneas 21 a 35), el funcionamiento del decodificador.

5.1.2 Programación mediante relación entradas/salidas

Por otro lado, en el listado 5.2 se programa el sistema de la figura 5.1b.

Observemos cómo nuevamente el programa supone dos procesos: el primero (líneas 11 a 19) programa el contador y el segundo (líneas 20 a 34) describe el decodificador.

```
1 library ieee;
2 use ieee.std_logic_1164.all ;
3 use work.s td_ari th.all ;
4 entity display1 is port(
5       clk,reset: in std_logic;
6       d: out std_logic_vector (6 downto 0));
7 end di spi ay 1;
8 architecture a_displ of displayl is
9 signal q: std_logic_vector (3 downto 0);
10 begin
11      process (clk,reset) begin
12         if (clk'event and clk = '1') then
13             q <= q + 1;
14                if (reset = '1' or q = "1001") then
15                   q <= "0000";
16                end if;
17
18          end if;
19       end process;
20      process (q) begin
21      case q is

23    when "0000" => d <= "1111110";
24    when "0001" => d <= "0110000";
25    when "0010" => d <= "1101101" ;
26    when "0011" => d <= "1111001" ;
27    when "0100" => d <= "0110011" ;
28    when "0101" => d <= "1011011" ;
29    when "0110" => d <= "1011111" ;
30    when "0111" => d <= "1110000" ;
31    when "1000" => d <= "1111111" ;
32    when "1001" => d <= "1110011" ;
33    when others => d <= "0000000";
33           end case ;
34      end process;
35 end a_displ;
```

Listado 5.2 Integración de las entidades utilizando señales.

Sin embargo, a diferencia del programa anterior en éste sólo se asignan las terminales de salida correspondientes al decodificador (línea 6), el enlace interno entre el contador y este último se realiza mediante señales (signal) línea 9.

De nuevo, es importante mencionar que los dos programas son correctos y que la disponibilidad de terminales y capacidad de integración del dispositivo empleado determinan la utilización de uno u otro. Por ejemplo, con el circuito CPLD CY372i de Cypress (apéndice D), el primer programa (Listado 5.1) utiliza el 9% de los recursos del dispositivo, mientras que el segundo (Listado 5.2) sólo utiliza 8 por ciento. Note la importancia del ahorro de recursos, que si bien en este caso no es significativo, al momento de realizar grandes diseños es importante, ya que permite ahorrar al máximo el espacio disponible dentro de los circuitos.

A continuación se presenta la información que reporta el porcentaje utilizado por los listados 5.1 y 5.2 compilados en el circuito CPLD CY372i de Cypress Semiconductor.

```
Information: Macrocell Utilization.
```

Description	Used	Max	
1 Dedicated Inputs 1	1	3	1
1 Clock/Inputs 1	1	2	1
1 I/O Macrocells 1	11	32	1
1 Buried Macrocells 1	0	32	1
1 PIM Input Connects 1	9	156	1

22 / 225 = 9%

Reporte de Listado 5.1

```
Information: Macrocell Utilization.
```

Description	Used	Max	
1 Dedicated Inputs 1	1	1 3	1
1 Clock/Inputs 1	1	1 2	1
1 I/O Macrocells 1	7	I 32	1
1 Buried Macrocells 1	4	1 32	1
1 PIM Input Connects 1	5	1 156	1

18 / 225 = 8 %

Reporte de Listado 5.2

5.2 Integración de entidades básicas

Con base en la descripción del apartado anterior y siguiendo con la filosofía de este libro, en relación con la integración de entidades no utilizaremos técnica alguna de programación específica; por el contrario, propondremos varias opciones de solución "no necesariamente la mejor" con objeto de que usted conozca más estrategias de programación y más palabras reservadas por VHDL. Esperamos que la experiencia adquirida le permita en el futuro desarrollar soluciones novedosas e ingeniosas.

5.2.1 Programación de tres entidades individuales

En la figura 5.2 se muestra un circuito lógico formado por los siguientes subsistemas: teclado, codificador, registro, decodificador de siete segmentos activo en bajo y un *display* de siete segmentos. La finalidad de este sistema electrónico es observar en el *display* el número decimal equivalente al de la tecla presionada.

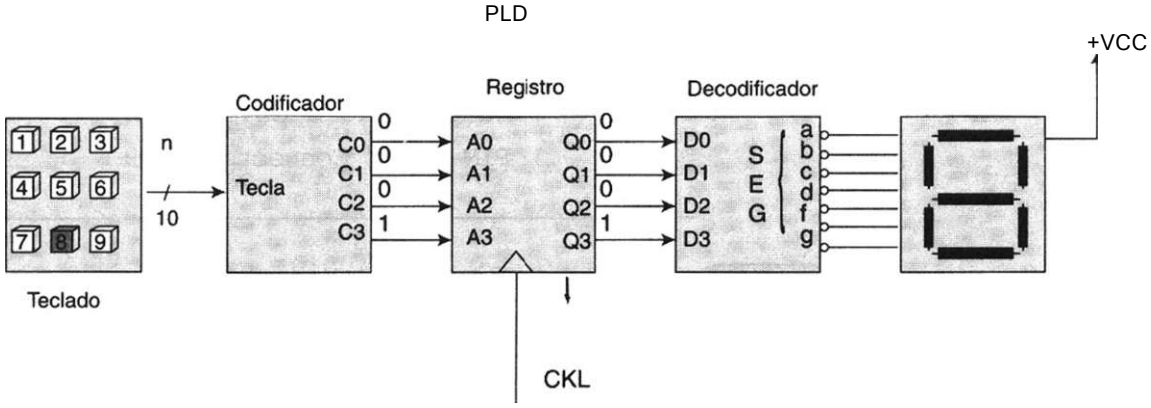


Figura 5.2 Sistema visualizador de un número en un display.

Para comenzar, el usuario presiona la tecla decimal correspondiente al número que se desea que aparezca en el *display*. El codificador convierte en código BCD el equivalente al número decimal presionado; por ejemplo, si se presiona la tecla correspondiente al número 8, el codificador entregará el valor BCD 1000, valor que luego se transfiere vía el registro hacia el decodificador de siete segmentos para observar en el display el valor del número decimal seleccionado.

La programación de estos módulos se muestra en el listado 5.3. Como podemos observar, se ha hecho de manera individual para cada bloque, por lo cual se han asignado terminales para cada uno de ellos (líneas 6, 7, 8 y 9). El bloque del codificador se encuentra programado de las líneas 14 a la 22, mientras que la programación del registro se muestra en las líneas 24 a la 28; por último, la parte del decodificador va de la línea 29 a la 40.

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity displayl is port(
4      clk: in std_logic;— pulso de reloj
5    tecla: in std_logic_vector (0 to 8) ;       — botón del teclado
6        C: inout std_logic_vector (3 downto 0);—salida codificador
7        A: inout std_logic_vector (3 dcwnto 0);— entrada registro
8        Q: inout std__logic_vector (3 downto 0);— salida registro
9        D: inout std_logic_vector (3 dcwnto 0);—entrada decodificador
10     seg: out std_logic_vector (0 to 6));  —salidas decodificador
11 end displayl;
12  architecture a_displ of displayl is
13   begin
14     C <= "0001" when tecla = "100000000" else
15          "0010" when tecla = "010000000" else
16          "0011" when tecla = "001000000" else
17          "0100" when tecla = "000100000" else
18          "0101" when tecla = "000010000" else
19          "0110" when tecla = "000001000" else
20          "0111" when tecla = "000000100" else
21          "1000" when tecla = "000000010" else
22          "1001";
23                 A <= C;
24     process (clk,A,D) begin
25           if (clk'event and clk = '1') then
26               Q <= A;
27               D <= Q;
28           end if;
29    case d is
30        when "0000" => seg <= "0000001";
31        when "0001" => seg <= "1001111";
32        when "0010" => seg <= "0010010";
33        when "0011" => seg <= "0000110";
34        when "0100" => seg <= "1001100";
35        when "0101" => seg <= "0100100";
36        when "0110" => seg <= "0100000";
37        when "0111" => seg <= "0001110";
38        when "1000" => seg <= "0000000";
39       when others => seg <= "0001100";
40     end case;
41    end process;
42 end a_displ;
```

Listado 5.3 Programación del sistema visualizador de un número.

- El porcentaje total de recursos utilizados en el CPLD CY372i cuando se compila el programa es de 21%, según se observa en la información de la figura 5.3.

```
Information: Macrocell Utilization.
                    Description          Used      Max

1 Dedicated Inputs   1     3       |     3
1 Clock/Inputs       1     2       |     2
1 I/O Macrocells     1    23       |    32
1 Buried Macrocells  1     0       |    32
1 PIM Input Connects  1    20       |   156
                                    48 / 225 =21 %
```

Figura 5.3 Porcentaje utilizado en el CPLD C y 372i

Las terminales utilizadas dentro del circuito se muestran en el siguiente reporte de asignación de terminales proporcionadas por el programa Warp R4 de Cypress Semiconductor encargado de la compilación del dispositivo (Fig. 5.4).

Device: c372i
Package: CY7C372L66JC

```
 1  :  GND
 2  :  Not Used
 3  :  Not Used
 4  :  Not Used
 5  :  Not Used
 6  :  Not Used
 7  :  Not Used
 8  :  Not Used
 9  :  Not Used
10  :  tecla_IBV 0
11  :  VPP
12  :  GND
13  :  elk
14  :  cl_3
15  :  a_3
16  :  ^
17  :  C"°
18  :  a_°
19  :  a"2
20  :  C"2
21  :  cJL
22  :  VCC
23  :  GND
24  :  seg 4
25  :  c_3
26  :  seg 6
27  :  seg 1
28  :  seg 3
29  :  seg 2
30  :  seg 5
31  :  Not Used
32  :  tecla IBV 1
33  :  tecla IBV_2
34  :  GND
35  :  tecla_IBV 3
36  :  seg 0
37  :  q_2
38  :  d_1
39  :  d3
40  :  ¿0
42  :  d~0
43  :  ¿1
44  :  VCC
```

Figura 5.4 Reporte de asignación de terminales del programa Warp R4.

5.2.2 Programación de entidades individuales mediante asignación de señales

Desde otro ángulo, imaginemos que los bloques de la figura 5.2. se pueden ver como un solo módulo (Fig. 5.5). En esta última figura se aprecia que las salidas de una entidad funcionan como las entradas de otra entidad. Esta forma de programación es posible siempre y cuando los bloques individuales se relacionen mediante señales internas.

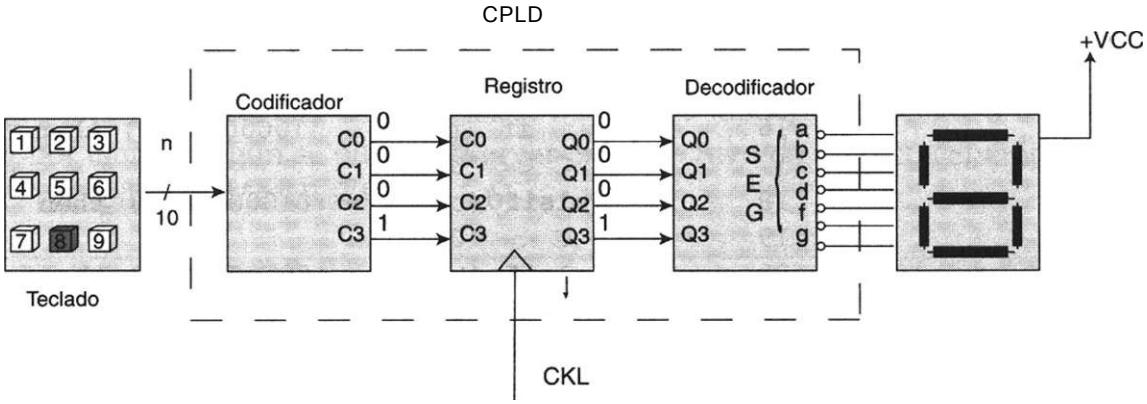


Figura 5.5 Sistema visualizador de un número utilizando señales internas.

Otro aspecto importante radica en que el circuito nada más recibe como entradas las líneas provenientes del teclado y como salidas las siete terminales del decodificador. Esta asignación es necesaria debido a que el dispositivo debe conectarse a elementos periféricos externos, tal es el caso del teclado y del *display.*

En el listado 5.4 se muestra la programación correspondiente a la integración de los submódulos del circuito de la figura 5.5.

```
1 library ieee;
2 use ieee.std_logic_l164.all;
3 entity display is port (
4       clk: in std_logic;
5       tecla: in std_logic_vector (0 to 8);—declaración entradas
6       seg: out std_logic_vector (0 to 6) );--declaración salidas
7 end display;
8 architecture a_disp of display is
9  signal C: std_logic_vector (3 downto OL-
IO signal Q: std_logic_vector (3 downto 0) ;
11   begin
12     process (clk,tecla,c,q) begin
13           if (clk1 event and clk - '1') then
14                     Q <= C;
15                end if;
16             if (tecla = "100000000") then
17           C <= "0001";
18          elsif (tecla = "010000000") then
19           C <= "0010";
20          elsif (tecla = "001000000") then
21           C <= "0011";
22          elsif (tecla = "000100000") then
23           C <= "0100";
24          elsif (tecla = "000010000") then
25           C <= "0101";
26          elsif (tecla = "000001000") then
27           C <= "0110";
28          elsif (tecla = "000000100") then
29           C <= "0111";
30          elsif (tecla = "000000010") then
31           C <= "1000";
32          else
33           C <= "1001";
34         end if;
35    case q is
36         when "0000" => seg <= "0000001";
37         when "0001" => seg <= "1001111";
38         when "0010" => seg <= "0010010";
39         when "0011" => seg <= "0000110";
40         when "0100" => seg <= "1001100";
41         when "0101" => seg <= "0100100";
42         when "0110" => seg <= "0100000";
43         when "0111" => seg <= "0001110";
44         when "1000" => seg <= "0000000";
45         when others => seg <= "0001100";
4 6  end case;
47     end process;
48  end a_disp;
```

Listado 5.4 Código VHDL del sistema visualizador utilizando señales internas.

Observe cómo en el programa (listado 5.4) se integraron las entidades del codificador, registro y decodificador de BCD a siete segmentos en una entidad llamada *display* (línea 3 a la 7). Note que únicamente se han declarado las entradas del teclado (tecla) y las salidas que se conectan al *display* de siete segmentos (seg). La descripción general del programa se basa en el cambio de asignación de variables que ilustra la figura 5.5, donde se puede ver que los bloques se interconectan por medio de sus señales internas (C y Q), líneas 9 y 10. En la línea 14 se asigna a Q el valor de la señal C (Q < = C), cuando sucede el hecho *clk'event and clk = '1'*, que no es sino la habilitación del registro. En la línea 35 se observa que para un valor determinado de Q se activa un bit del vector de salida denominado *seg,* que presenta la información en el *display* de siete segmentos.

Al igual que en el caso anterior, el ahorro de terminales es significativo; además, la utilización del circuito se reduce de manera significativa de 21 a 9 por ciento, según se puede observar en el archivo de reporte generado por el compilador WARP R4 (Fig. 5.6).

Information: Macrocell Utilization.

Description	Used	Max	
1 Dedicated Inputs	1	3	1
1 Clock/Inputs	1	2	1
1 I/O Macrocells	1	32	1
! Buried Macrocells	1	32	1
1 PIM Input Connects	1	156	1

22 / 225 = 9 %

Figura 5.6 Archivo de reporte del compilador WARP R4.

Ejemplo 5.1 Realice la programación en VHDL para integrar los diferentes bloques lógicos mostrados en la figura E5.1.

Descripción. El circuito de la figura E5.1 se utiliza para automatizar el proceso de empaquetamiento de muñecas de porcelana. Considere que las muñecas se pueden empaquetar en forma individual o con un máximo de nueve unidades por paquete.

En un inicio el operador selecciona mediante el teclado decimal la cantidad de piezas que va a empaquetar. Como sabemos, este número decimal se convierte a código BCD a través del circuito codificador y luego pasa mediante el registro hacia el decodificador de siete segmentos para mostrar

en el *display* el valor del número seleccionado. Observe que este valor binario es a su vez la entrada A (A3, A2, Ai *y* A*0)* del circuito comparador de cuatro bits.

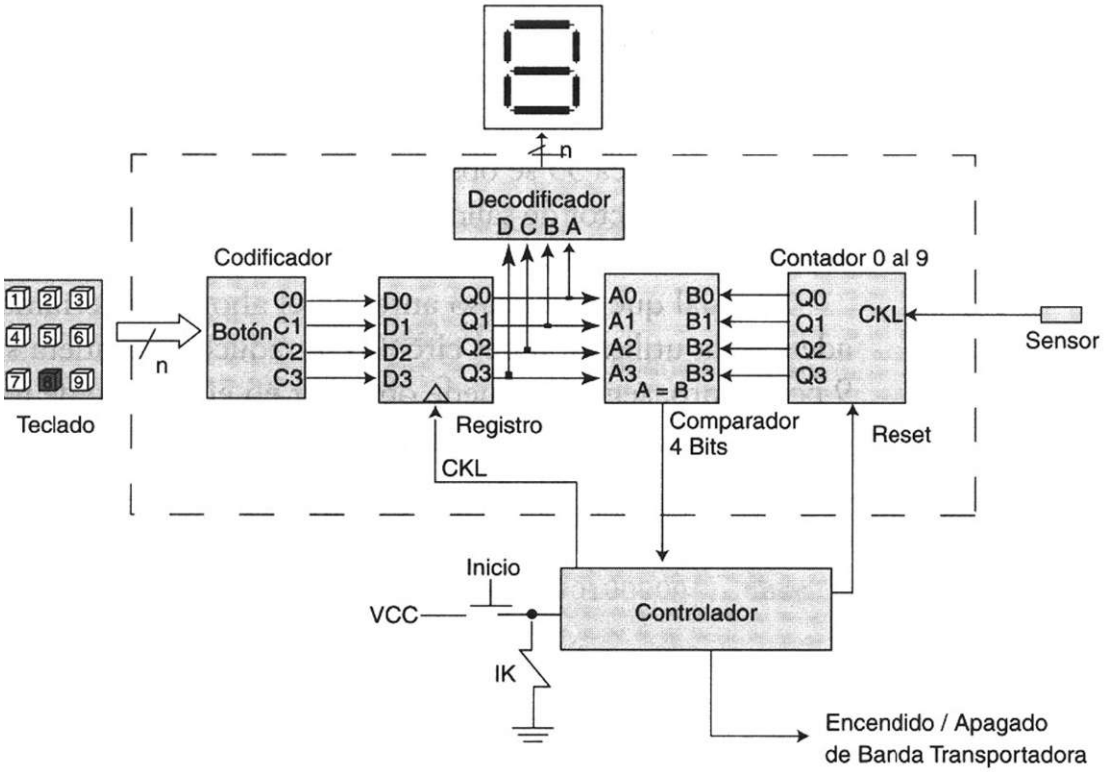


Figura E5.1 Automatización del sistema empaquetador de muñecas.

Después de seleccionar la cantidad de muñecas por empaquetar, el operador presiona el botón de inicio, que desencadena una serie de acciones controladas por el bloque denominado controlador. Comienza con una señal de salida llamada Reset, que coloca al contador binario en el estado de cero; enseguida envía la señal de arranque (Encendido) al motor que controla el avance de la banda transportadora.

Ahora bien, cada vez que una de las muñecas colocadas sobre la banda transportadora pasa por el "sensor" (Fig. E5.2), se origina un impulso eléctrico (pulso) que hace que el contador aumente en uno su conteo. Este procedimiento continúa incrementando al contador en una unidad, hasta el momento en que la cifra actual del contador (número B) es igual al "número A" dentro del comparador (A = B), con lo cual este último envía una señal al bloque controlador que detiene la banda transportadora (Apagado) y marca el fin del proceso.

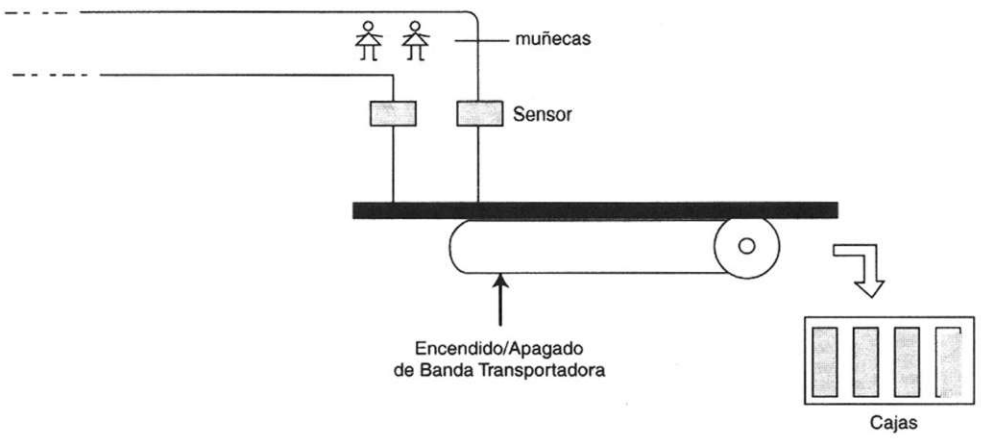


Figura E5.2 Sistema de sensado de muñecas.

En la figura E5.3 se observa el intercambio de señales que se realiza entre el controlador y cada uno de los bloques lógicos del sistema.

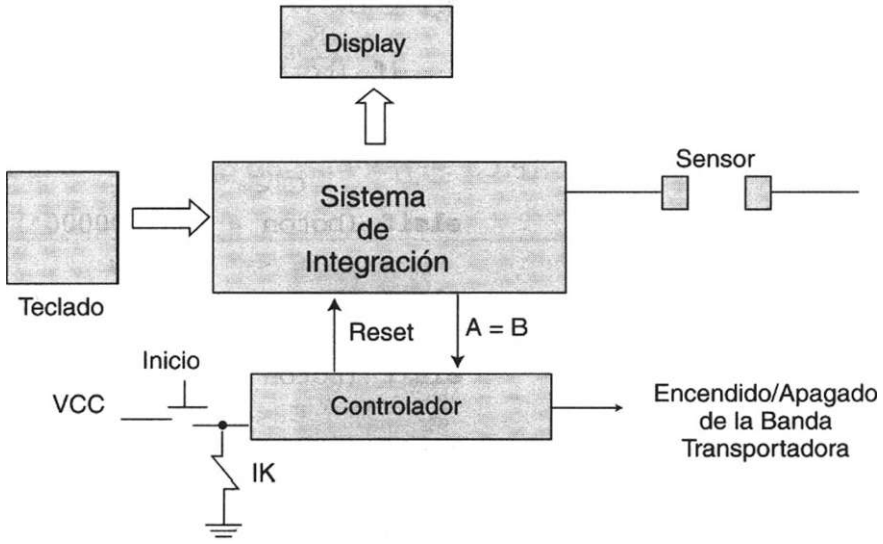


Figura E5.3 Interacción de las señales del controlador.

Solución

La programación correspondiente a esta integración se muestra en el listado E5.1.

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use work.std_arith.all;
4 entity control is port (
5     elk,reset: in std_logic;
6       boton: in std_logic_vector (0 to 8) ;
7      sensor: in std_logic;
8        deco: out std_logic_vector (0 to 6);
9     compara: out std_logic);
10 end control;
11 architecture a_control of control is
12 signal Q,C,R: std_logic_vector (3 downto 0);
13   begin
14     procesol: process (sensor,reset,q)begin
15         if (sensor'event and sensor = '1') then
16             Q <= "0000";
17             Q <= Q + 1;
18           if (reset = '1' or Q = "1001") then
19            Q <= "0000";
20               end if;
21          end if;
22 end process;
23        proceso2: process (elk,boton,R) begin
24        if (elk'event and elk = '1') then
25                 R <= C;
26            if (boton = "100000000") then
27                   C <= "0001";
28         elsif (boton = "010000000") then
29                   C <= "0010";
30         elsif (boton = "001000000") then
31                   C <= "0011";
32         elsif (boton = "000100000") then
33                   C <= "0100";
34         elsif (boton = "000010000") then
35                   C <= "0101";
36         elsif (boton = "000001000") then
37                    C <= "0110";
38         elsif (boton = "000000100") then
39                   C <= "0111";
40         elsif (boton = "000000010") then
41                   C <= "1000";
42         else
43                   C <= "1001";
44         end if;
45
46     case R is
47            when "0000" => deco <= "0000001";
48            when "0001" => deco <= "1001111";
```

Continúa

```
49              when "0010" => deco <= "0010010";
50              when "0011" => deco <= "0000110";
51              when "0100" => deco <= "1001100";
52              when "0101" => deco <= "0100100";
53              when "0110" => deco <= "0100000";
54              when "0111" => deco <= "0001111";
55              when "1000" => deco <= "0000000";
56              when others => deco <= "0001100";
57          end case ;
58  end process;
59
60      compara <= '1' when Q = R else '0';
61
62  end architecture;
```

Listado E5.1 Programación del sistema visualizador de un número.

Observe que la integración de las entidades que forman el sistema sólo precisó dos procesos: el primero (línea 14), para describir el conteo de las muñecas mediante la variable de entrada sensor; el segundo (línea 23), para integrar el codificador, registro y decodificador de siete segmentos. Vea también que en la línea 60 se asigna a la variable compara el valor de uno cuando el valor binario de los vectores Q y R son iguales. Esto con objeto de detener la banda transportadora.

Ejemplo 5.2 Consideremos ahora el caso en que el número máximo de muñecas que se puede empaquetar pasó de 9 a 99 (Fig. E5.4).

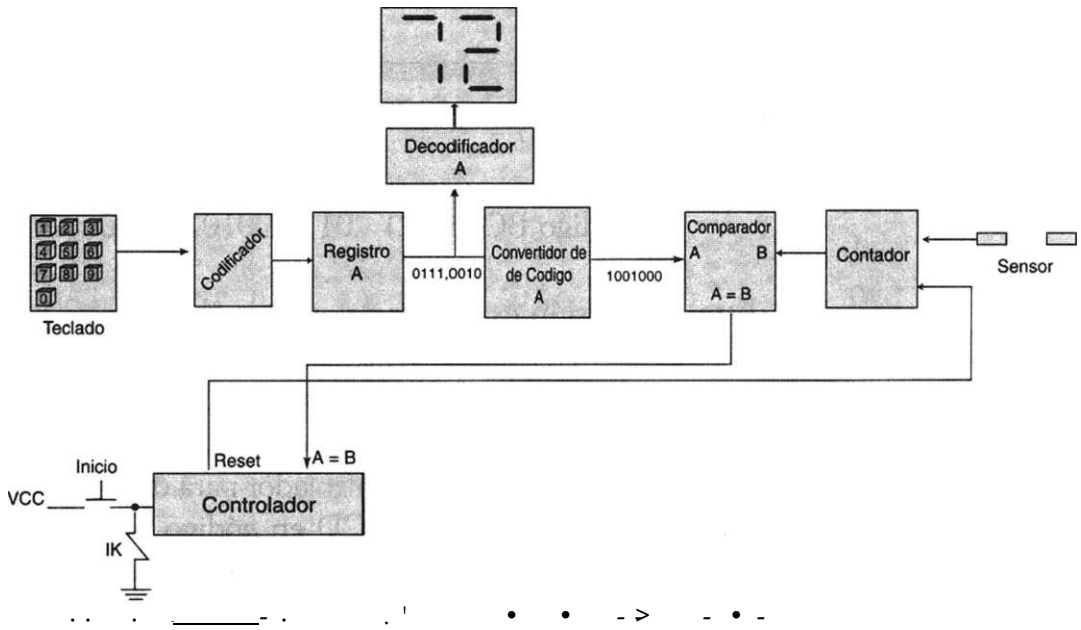


Figura E5.4 Sistema visualizador del 0 al 99.

Al igual que en el ejemplo anterior, el operador selecciona mediante el teclado decimal la cantidad de muñecas por empaquetar. Como sabemos, este número decimal se convierte en código BCD mediante el circuito codificador y luego pasa vía el registro hacia los decodificadores de siete segmentos para observar en el *display* el valor del número seleccionado. Sin embargo y a diferencia del circuito anterior, el operario ahora puede elegir un número de dos dígitos; como ejemplo, supongamos que teclea el número decimal 72 mediante la pulsación del 7 y luego del 2. Estos números codificados en BCD proporcionan el siguiente código:

BCD (7) =0111 BCD (2) = 0010

Como puede observar, el vector de entrada es de 8 bits (0111 0010). Este valor pasa vía el registro a los decodificadores de siete segmentos para visualizar en el *display* las cifras correspondientes a este valor. Note que el número BCD (72) es introducido en un nuevo bloque denominado convertidor de código que tiene como función convertir código BCD en código binario (Fig. E5.5).

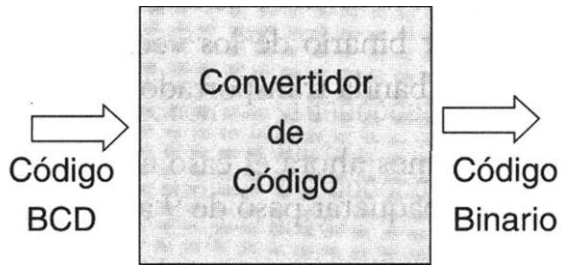


Figura E5.5 Convertidor de código.

Código BCD (72) 0111,0010 = 1001000 (binario)

Esta conversión es necesaria, ya que el contador incrementa su valor en forma binaria cada vez que sensa una muñeca. Con esto se intuye que la función del comparador es relacionar el valor binario proveniente del teclado y el valor que proporciona el contador; cuando estos números son iguales (A = B) se envía un pulso al controlador para detener la banda transportadora.

La conversión de código BCD en código binario puede realizarse a través de circuitos sumadores. Básicamente, para obtener un número binario a partir de un número BCD hay que sumar los números binarios que representan los pesos de los bits del número BCD.

Por ejemplo, considere que tenemos el número BCD (72).

B3	B2	Bl	BO	A3	A2	A1	AO
0	1	1	1	0	0	1	0
	7				2		

El grupo de 4 bits más a la izquierda representa 70, y el grupo de 4 bits más a la derecha, el número 2. Esto quiere decir que el grupo más a la izquierda tiene un peso de 10 y el de la extrema derecha un peso de 1. En cada grupo, el peso binario de cada bit es el siguiente:

	Dígito de las decenas				Dígito de las unidades			
Peso:	80	40	20	10	8	4	2	1
Designación de bit:	B_3	b_2	B_1	B_0	A_3	A2	A!	AQ

Por otro lado, el equivalente binario de cada bit BCD es un número binario que representa el peso de cada bit dentro del número BCD completo.

Por ejemplo, en la tabla E5.1 se muestra que el valor del bit BCD A0 tiene un peso de uno, por lo cual su valor binario representado por 7 bits es 0000001. De igual manera, el bit BCD B3 tiene un peso de 80, por lo que el valor binario correspondiente representado en 7 bits es 1010000. Observe que este valor se obtiene mediante la suma de (64 + 16) en notación binaria.

Bit	Peso	Representación binaria						
BCD	BCD	64	32	16	8	4	2	1
A0	1	0	0	0	0	0	0	1
A1	2	0	0	0	0	0	1	0
A2	4	0	0	0	0	1	0	0
A3	8	0	0	0	1	0	0	0
B0	10	0	0	0	1	0	1	0
B1	20	0	0	1	0	1	0	0
B2	40	0	1	0	1	0	0	0
B3	80	1	0	1	0	0	0	0

Tabla E5.1 Valores y pesos de los códigos BCD y binario.

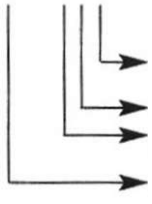
Para reafirmar el procedimiento de conversión BCD en binario veamos los siguientes ejercicios resueltos.

Convertir en binario los números
a) BCD 00100111 (27 decimal)
b) BCD 10011000 (98 decimal).

Solución

a) Primero se escribe el número BCD correspondiente y enseguida se suma el peso de las unidades y decenas.

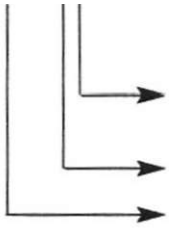
Valor BCD
00100111



0000001	1
0000010	2
0000100	4
0010100	20
0011011	número binario que representa al decimal 27

b)

Valor BCD
10011000



0001000	8
0001010	10
1010000	80
1100010	número binario que representa el decimal 98

Después de observar el procedimiento de conversión, podemos construir un circuito lógico mediante sumadores que realicen la suma correspondiente a los unos necesarios para la transformación de un número BCD en binario.

En la tabla E5.1 se puede observar que la columna "1" (bit menos significativo) de la representación binaria tiene un solo 1 y no existe la posibilidad de entrada de acarreo, de forma que basta tener una conexión directa desde el bit A0 de la entrada BCD a la salida binaria menos significativa (Fig. E5.8).

En la columna "2" de la representación binaria debe realizarse la suma de los bits Al y B0 del número BCD. Esta suma puede efectuarse a través de un circuito medio sumador como vimos anteriormente en el capítulo 3 Fig. E5.6).

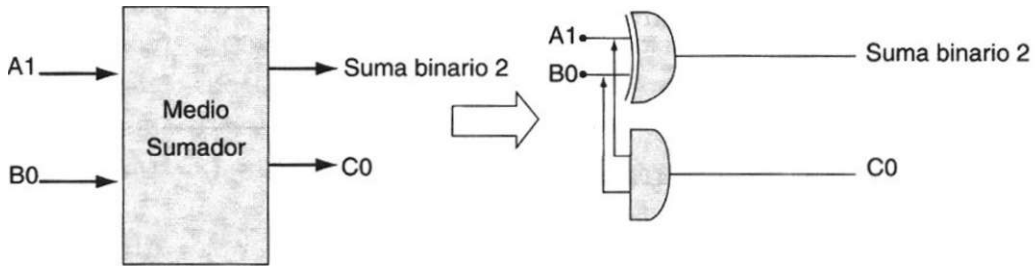


Figura E5.6 Medio sumador en bloque y con compuertas.

En la columna "4" de la representación binaria, la posible ocurrencia de dos unos ocurre sumando los bits A2 y Bl del número BCD y el acarreo de la suma anterior. Con esto, la suma tiene lugar a través de un sumador completo (Fig. E5.7).

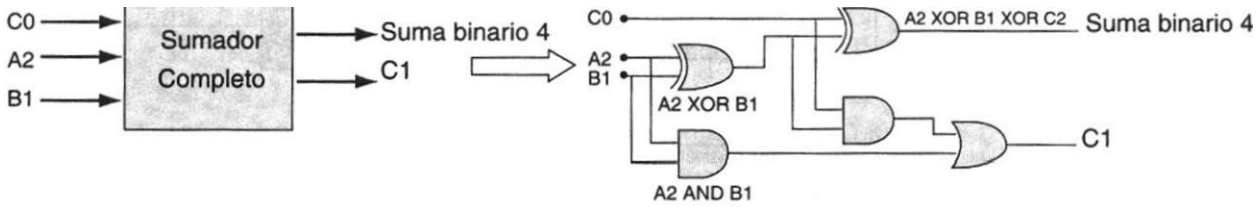


Figura E5.7 Sumador completo en bloque y con compuertas.

Con esta lógica podemos intuir el comportamiento del circuito de la figura E5.4. En resumen, en la columna "8" en la representación binaria aparecen tres unos, por lo que sumamos los bits A3, BO y B2 del número BCD. En la columna "16", sumamos los bits Bl y B3. En la columna 32 sólo es posible un uno único, por lo que se suma el bit B2 al acarreo de la columna "16". En la columna "64", sólo puede aparecer un uno único, de manera que el bit B3 se suma al acarreo de la columna "32".

El sistema sumador que realiza esta conversión de BCD a binario se muestra en la figura E5.8.

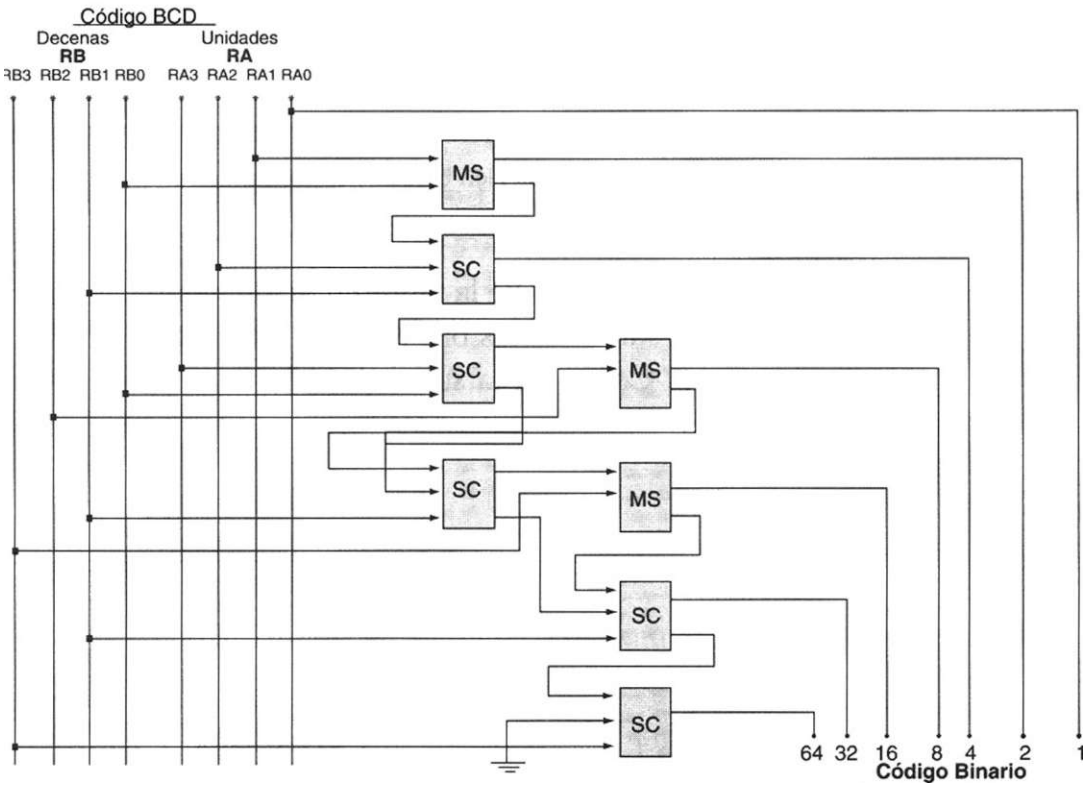


Figura E5.8 Circuito convertidor de código BCD en binario.

Después de analizar el proceso de conversión de BCD en binario, podemos describir el programa VHDL que integra las entidades de la figura E5.4 (listado E5.2).

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use work.std_arith.all;
4
5 entity control99 is port(
6       reset: in std_logic; —señal proveniente del controlador
7       boton: in std_logic_vector (0 to 9);—teclado
8      sensor: in std_logic; —señal que sensa el paso de una muñeca
9 dispa,dispb: out std_logic_vector (0 to 6); — display doble (a y b)
10    compara: out std_logic);
11 end control99;
12
13 architecture a_control99 of control99 is
```

Continúa

```
15 signal C,RA,RB: std_logic_vector (3 downto 0);— cod. y registro
16 signal Q,CONV: std_logic_vector (6 downto 0);—cont. y convertidor
17 signal tecla: std_logic_vector (1 downto 0);-- teclas presionadas
18 signal carry: std_logic_vector (0 to 6); — bit de acarreo del conv
19 signal x: std_logic_vector (0 to 1) ; — señal interna del conv
20
21 begin
22         teclado: process (boton,C,TECLA)
23          variable i: std_logic_vector (1 downto 0):= "00";
24            begin
25               if (boton = "1000000000") then
26                     C <= "0000";tecla <= i;
27              elsif (boton = "0100000000") then
28                     C <= "0001";tecla <= i;
29              elsif (boton = "0010000000") then
30                     C <= "0010";tecla <= i;
31              elsif (boton = "0001000000") then
32                     C <= "0011";tecla <= i;
33              elsif (boton = "0000100000") then
34                     C <= "0100";tecla <= i;
35              elsif (boton = "0000010000") then
36                     C <= "0101";tecla <= i;
37              elsif (boton = "0000001000") then
38                     C <= "0110";tecla <= i;
39              elsif (boton = "0000000100") then
40                     C <= "0111";tecla <= i;
41              elsif (boton = "0000000010") then
42                     C <= "1000";tecla <= i;
43              else
44                     C <= "1001";tecla <= i;
45              end if;
46
47                     if (tecla = "00") then
48                            RB <= C;
49                     else
50                            RA <= C;
51                     end if;

53                     i := i+1;

55              if (i = "10") then
56                    i := "00";
57              end if;

59       end Drocess :
```

Continúa

```
61 display:process (RA,RB) begin
62      case RA is —enciende display menos significativo
63            when "0000" => dispa <= "0000001";
64            when "0001" => dispa <= "1001111";
65            when "0010" => dispa <= "0010010";
66            when "0011" => dispa <= "0000110";
67            when "0100" => dispa <= "1001100";
68            when "0101" => dispa <= "0100100";
69            when "0110" => dispa <= "0100000";
70            when "0111" => dispa <= "0001111";
71            when "1000" => dispa <= "0000000";
72            when others => dispa <= "0001100";
73          end case ;
74
75       case RB is —enciende display mas significativo
76            when "0000" => dispb <= "0000001";
77            when "0001" => dispb <= "1001111";
78            when "0010" => dispb <= "0010010";
79            when "0011" => dispb <= "0000110";
80            when "0100" => dispb <= "1001100";
81            when "0101" => dispb <= "0100100";
82            when "0110" => dispb <= "0100000";
83            when "0111" => dispb <= "0001111";
84            when "1000" => dispb <= "0000000";
85            when others => dispb <= "0001100";
86          end case;
87
88  end process;
89
90     convertidor: process ( RA,RB,CARRY,X )
91      variable z: std_logic := '0';
92       begin
93         CONV(O) <= RA(0) ;
94         CONV(l) <= RA(1) xor RB(0);
95        CARRY(0) <= RA(1) and RB(1);
96         CONV(2) <= RA(2) xor RB(1) xor CARRY(0) ;
97        CARRY(1)<= (RA(2) and RB(1)) or (CARRY(0) and (RA(2) xor
98                   RB(1))) ;
99            X(0) <= RA(3) xor RB(0) xor CARRY(1) ;
100       CARRY(2) <= (RA(3) and RB(0)) or (CARRY(1) and(RA(3)
101                  xor RB(01 ) ) ;
102        CONV(3) <= X(0) xor RB(2);
103       CARRY(3) <= X(0) and RB(2);
104           X(l) <= CARRY(2) xor RB(1) xor CARRY(3);
105       CARRY(4) <= (CARRY(2) and RB(l))or (CARRY(3) and (CARRY(2)
106                xor RB(1) ) ) ;
```

Continúa

```
107         CONV(4) <= X(1) xor RB(3) ;
108         CARRY(5)<= X(1)and RB(3);
109         CONV(5) <= CARRY(4) xor RB(1) xor CARRY(5);
110         CARRY(6)<= (CARRY(4) andRB(1)) or(CARRY(5) and (CARRY(4)
111                    xor RB(1))) ;
112         CONV(6) <= ( z xor RB(3) xor CARRY(6) ) ;
113
114  end process;
115
116  contador: process (sensor,reset,Q) begin
117         if (sensor' event and sensor = '1') then
118             Q <= "0000000";
119              Q <= Q + 1;
120
121             if (reset = '1' or Q = "1100011") then --reset en 99
122                   Q <= "0000000";
123              end if;
124         end if;
125  end process;
126
127   --SALIDA DEL COMPARADOR (A = B)
128
129             compara <= ' 1' when Q = CONV else ' 0 ' ;
130 end architecture;
```

Listado E5.2 Programación del sistema visualizador del 0 al 99.

Ahora analicemos el programa anterior. En las líneas 5 a 11 se declaró la entidad de diseño denominada control 99. En ésta se han declarado los siguientes bloques lógicos: codificador, registro, convertidor, comparador, decodificador y contador.

En la línea 15 se declararon las señales utilizadas en el diseño. El significado de cada una se explica en la tabla E5.2.

Señal	Descripción
C	Señales de salida del codificador y de entrada al registro
RA,RB	Señales de salida del registro. RA representa los cuatro bits menos significativos y RB los más significativos.
Q	Señal de salida del contador y de entrada al comparador
CONV	Señal de salida del convertidor de código y de entrada al comparador
CARRY	Señal interna del módulo convertidor de código. Funciona como acarreo interno.
X	Señal interna auxiliar en el desarrollo del convertidor de código

Tabla E5.2 Significado de las señales utilizadas en el diseño.

En lo que respecta a la arquitectura del programa, observe que se estructuró utilizando cuatro procesos en los cuales se detalla cada módulo del sistema; por ejemplo para el codificador y el teclado tenemos un proceso llamado teclado (línea 22), que captura el dato proveniente del teclado, lo codifica y detecta si la tecla se presionó por primera o segunda vez; luego, estos datos se transfieren al registro. En el proceso llamado display (línea 61) se decodifica la entrada proveniente del registro y se asigna la salida correspondiente al dígito seleccionado. De acuerdo con esta secuencia, el siguiente proceso convertidor (línea 90) se realiza con la finalidad de desarrollar el módulo que convierte BCD en código binario (Fig. E5.8).

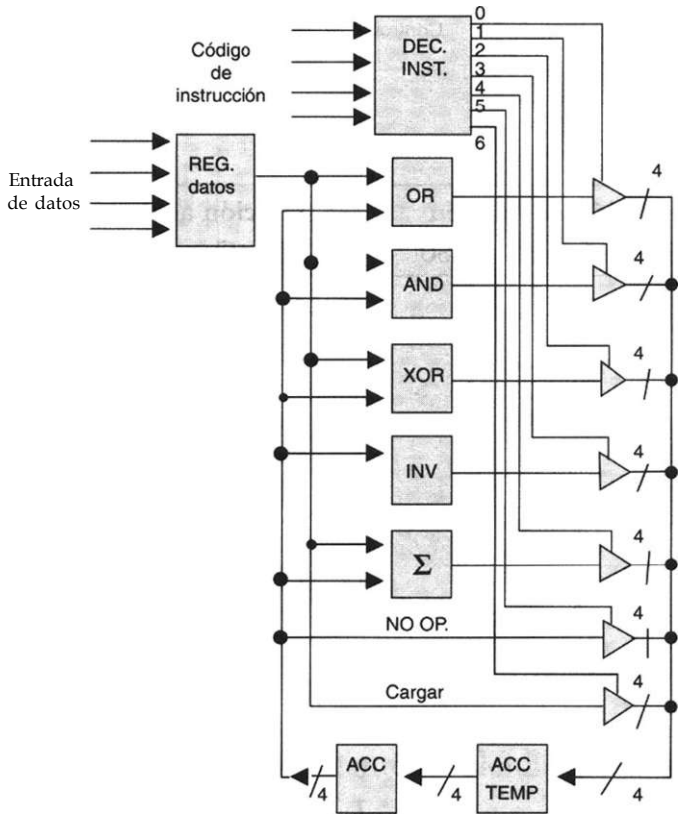
Por último, el proceso contador (línea 116) genera la cuenta del 0 al 99 incrementándose en una unidad cada que el sensor detecta un objeto. Ya fuera del proceso, en la línea 129 se compara el vector proveniente del convertidor (CONV) con la salida del contador (Q). Si son iguales, el controlador envía un pulso para detener la banda transportadora.

Ejercicios

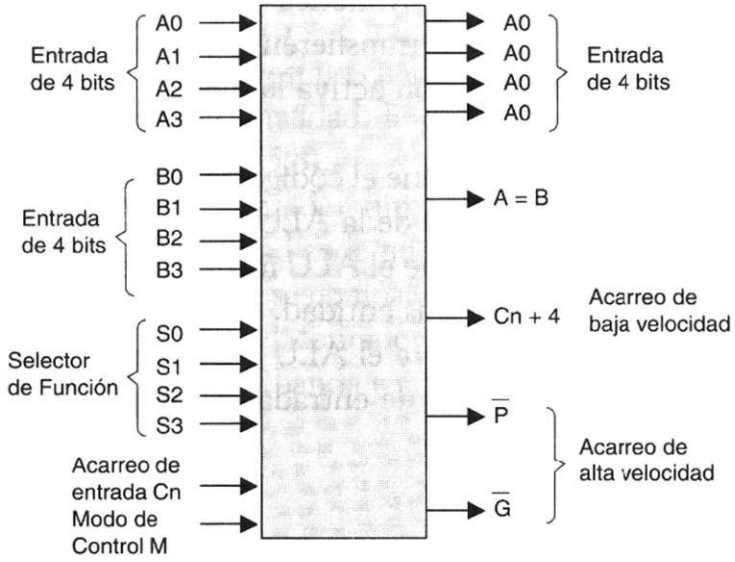
Integración de entidades

5.1 Considere el diagrama mostrado en la siguiente figura que corresponde a una unidad aritmética y lógica (ALU) de 4 bits. Como puede observarse los datos se transfieren a través del registro de datos y el decodificador de instrucción activa la función que se realiza con estos datos.

a) Determine el código de instrucción que se utiliza para activar las operaciones de la ALU.
b) Programe el ALU mediante el uso de varios procesos para adherir entidad tras entidad.
c) Considere el ALU como un sistema general y programe mediante la relación de entradas y salidas.



5.2 En la siguiente figura se muestra el circuito electrónico de una unidad aritmética y lógica. Considere la tabla de funcionamiento mostrada en el ejercicio y programe utilizando la integración de entidades mediante la relación de señales de entrada - salida.

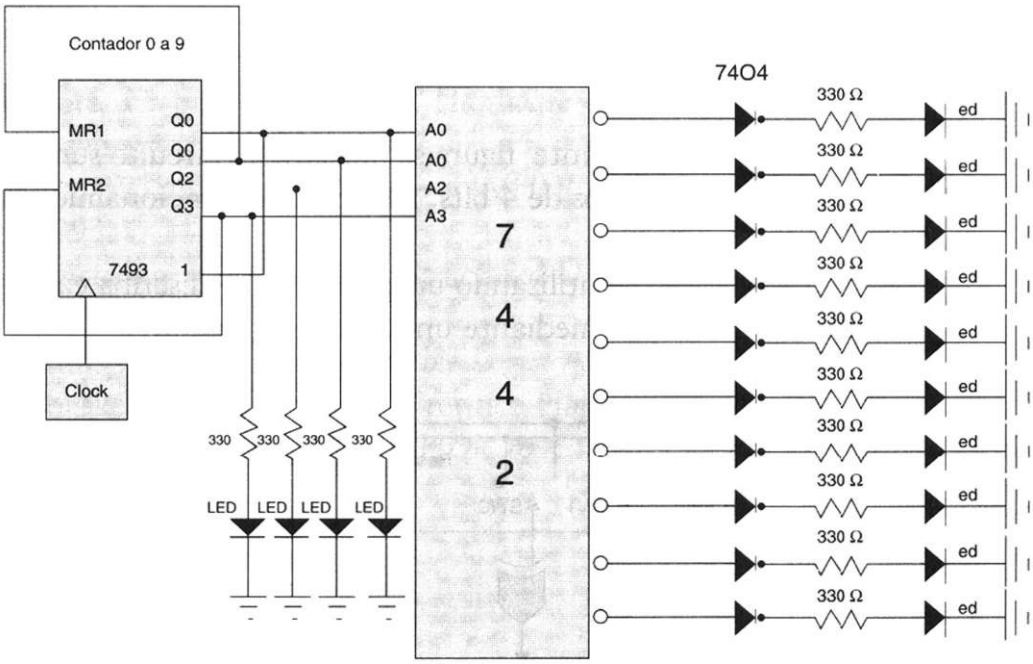


Selección de función				Función aritmética de salida	Función lógica de salida
S3	**S2**	SI	so	**M = Cn = 0**	**M = 1**
0	0	0	0	F = A	F = /A
0	0	0	1	F= A + B	F = /(A + B)
0	0	1	0	F = A + /B	F = /A • B
0	0	1	1	F = -1 (complemento a 2)	F = 0000
0	1	0	0	F = A • (A • /B)	F = /(A • B)
0	**1**	0	**1**	F = (A + B) • (A /B)	F = /B
0	**1**	**1**	0	F = A-B-l	F = A B
0	1	**1**	**1**	F = A • /B -1	F = A • /B
1	0	0	0	F = A • (A -B)	F = /A + B
1	0	0	**1**	F = A • B	F = /(A B)
1	0	**1**	0	F = (A + /B) • A • B	F = B
1	0	**1**	1	F = A * B - 1	F = A • B
1	1	0	0	F = A • (A = 2A)	F = 1111
1	**1**	0	1	F = (A + B) • A	F = A + /B
1	**1**	**1**	0	F = (A + /B) • A	F = A + B
1	**1**	**1**	1	F = A - 1	F = A

5.3 En la siguiente figura se muestra un circuito contador y un decodificador de binario a decimal ,código BCD , cuya función consiste en activar cada una de las salidas decimales del decodificador mediante el valor del binario correspondiente procedente del contador. Así por ejemplo si el contador se encuentra en el estado cero (0000) la salida del decodificador que se activa es la SO. Observe cómo el decodificador tiene las salidas activas en bajo, por lo que para encender los led de la figura se requiere el uso de un inversor.

Realice un programa en VHDL que sustituya al contador, decodificador e inversores.

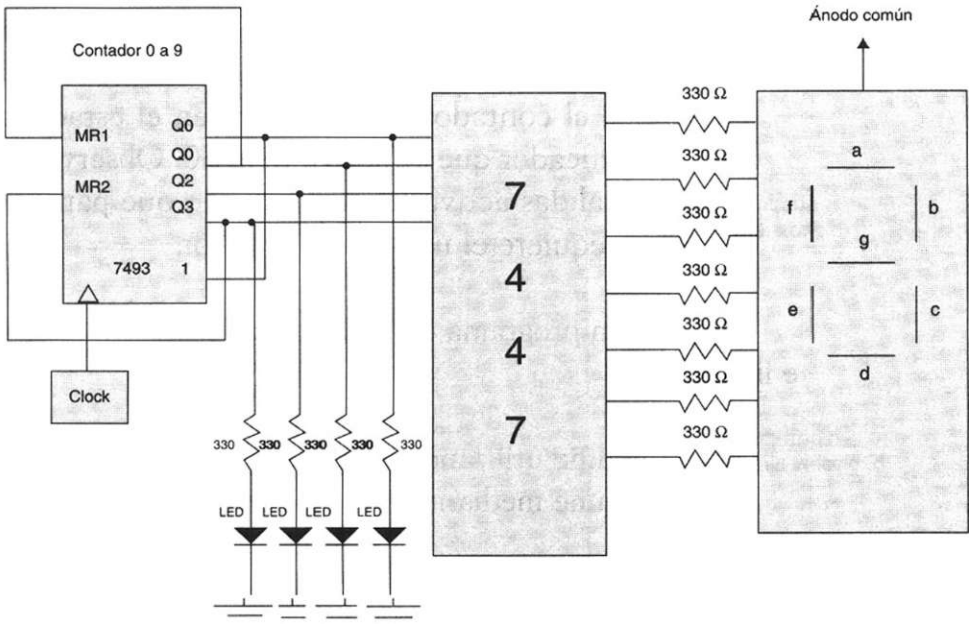
a) Programe utilizando un proceso que describa cada entidad de diseño
b) Programe mediante un algoritmo general



5.4 En la siguiente figura se muestra un circuito contador y un decodificador de 7 segmentos, la función del circuito consiste en desplegar en el display el valor decimal correspondiente al valor binario originado en el contador.

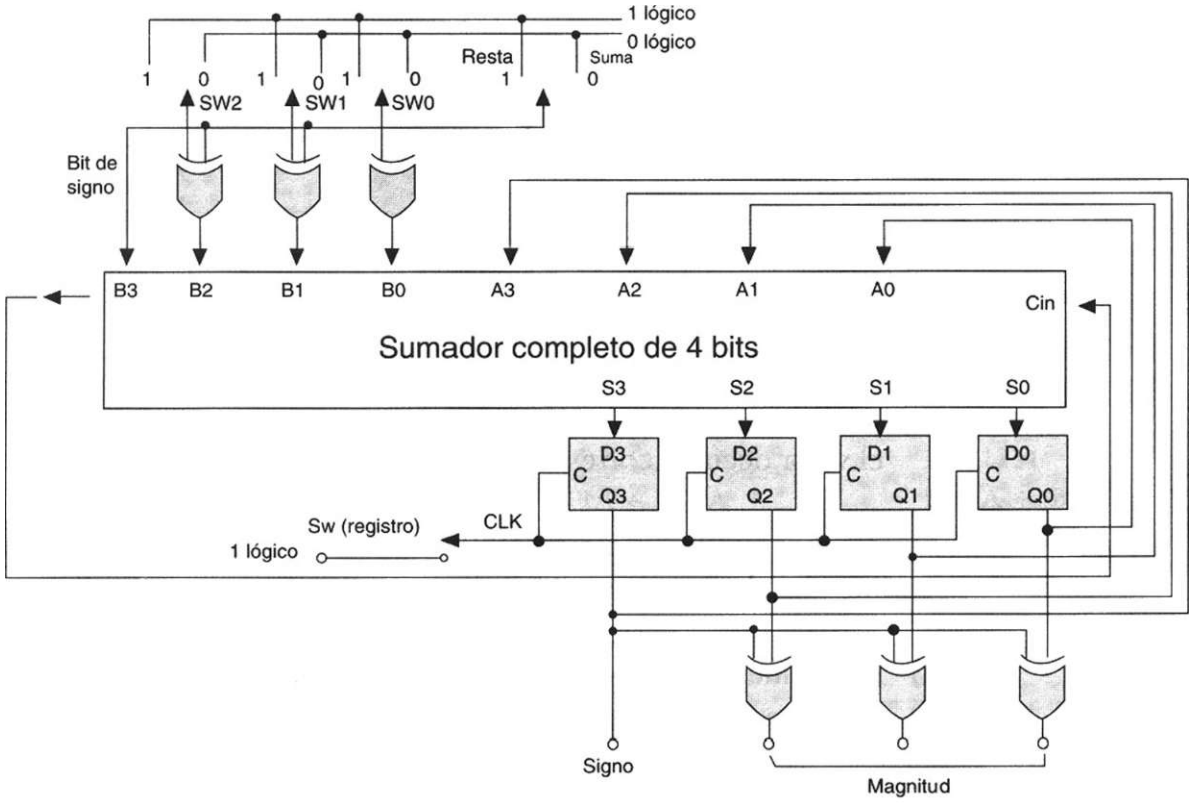
Realice un programa en VHDL que sustituya al circuito contador y decodificador de 7 segmentos.

a) Programe utilizando un proceso que describa cada entidad de diseño
b) Programe mediante un algoritmo general



5.5 En la siguiente figura se muestra un circuito sumador-restador de números signados de 4 bits. Interprete el funcionamiento del sistema y:

a) Programe utilizando un proceso que describa cada entidad de diseño
b) Programe mediante un algoritmo general



Bibliografía

T.L. Floyd. *Fundamentos de sistemas digitales.* Prentice Hall, 1998.

Ll. Teres. Y. Torroja. S. Olcoz; E. Villar. *VHDL, lenguaje estándar de diseño electrónico.* McGraw-Hill, 1998.

David G. Maxinez, Jessica Alcalá. *Diseño de Sistemas Embebidos a través del Lenguaje de Descripción en Hardware VHDL.* XIX Congreso Internacional Académico de Ingeniería Electrónica. México, 1997.

C. Kloos, E. Cerny. *Hardware Description Language and their Aplications. Specification, Modelling, Verification and Synthesis of Microelectronic Systems.* Chapman & Hall, 1997.

IEEE: *The IEEE standard VHDL Language Reference Manual.* IEEE-Std-1076-1987, 1988.

Zainalabedin Navabi. *Analysis and Modeling of Digital Systems.* McGraw-Hill, 1988.

P J. Ashenden. The *Designer's guide to VHDL.* Morgan Kauffman Publishers, Inc., 1995.

R. Lipsett, C. Schaefer. *VHDL Hardware Description and Design.* Kluwer Academic Publishers, 1989.

S. Mazor, P Langstraat. A *guide to VHDL.* Kluwer Academic Publishers, 1993.

J.R. Armstrong y F. Gail Gray. *Structured Design with VHDL.* Prentice Hall, 1997.

K. Skahill. *VHDL for Programmable Logic.* Addison Wesley, 1996.

J. A. Bhasker. A *VHDL Primer.* Prentice Hall, 1992.

H. Randolph. *Applications of VHDL to Circuit Design.* Kluwer Academic Publisher.

La secuencia de acciones que realiza un controlador puede describirse mediante los diagramas de estado que se vieron en el capítulo 4, o a través de un diagrama diseñado específicamente para definir algoritmos de hardware denominado carta ASM (siglas en inglés de algoritmo de la máquina de estado). La ventaja principal de una carta ASM con respecto a los diagramas de estado es que permite controlar y especificar el flujo de la información al mismo tiempo.

Hay que recordar que un algoritmo describe paso a paso del comportamiento de un sistema, tomando en consideración las siguientes características:

- El algoritmo debe de ser finito. Debe tener un número determinado de estados.
- Tiene que ser definido. En cada estado deben establecerse por completo todas las acciones que se llevan a cabo. Esto incluye las entradas, salidas y decisiones que conducen a ese estado.

6.1 Algoritmo de la máquina de estado (ASM)

El algoritmo o carta ASM utiliza tres símbolos básicos para describir el comportamiento de un sistema:

- Bloque de estado
- Bloque de decisión
- Bloque de salida condicional

Bloque de estado

El bloque de estado representa el "estado" de una máquina secuencial y debe contener la siguiente información (Fig. 6.1).

- Nombre del estado. Por lo general se utilizan números (0,1,2,3,... etc.) o letras (A,B,C,... etc).
- Código del estado ("xxxx"). Se refiere al código binario asignado al estado.
- Lista de salidas. Señales de salida asignadas al estado y que sólo se encuentran activas durante el tiempo que permanezca el sistema en ese estado.

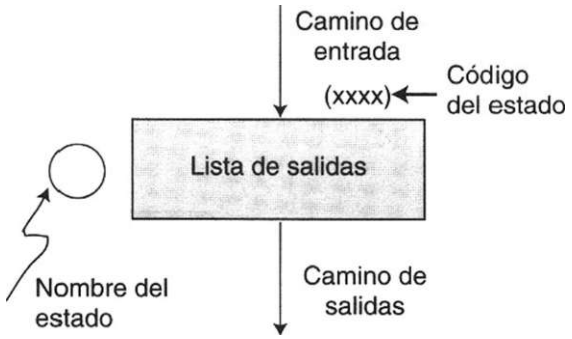


Figura 6.1 Descripción de un bloque de estado.

Bloque de decisión

El rombo o bloque de decisión se refiere a las variables de entrada al sistema y contienen la siguiente información (Fig. 6.2).

- Una variable de entrada. En este rombo se indica el nombre de la variable de entrada.
- Una salida verdadera.
- Una salida falsa.

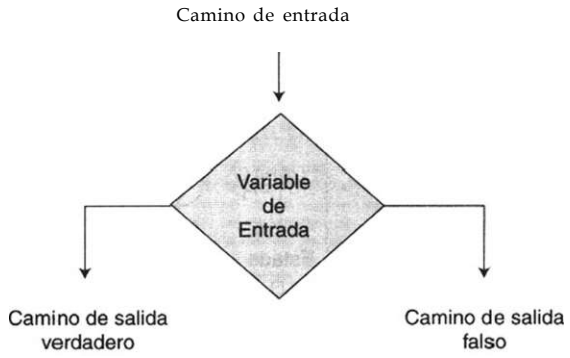


Figura 6.2 Descripción del bloque de decisión.

Bloque de salidas condicionales

El bloque de salidas condicionales (Fig. 6.3) se utiliza para activar señales de salida que sólo se encuentran disponibles para ciertas condiciones de entrada. La información contenida en dicho bloque es la siguiente:

- Una lista de salidas condicionales que dependen de cierta condición de entrada.

Figura 6.3 Descripción de un bloque de salidas condicionales.

6.2 Estructura de una carta ASM

Una carta ASM consiste de uno o más bloques ASM interconectados de una manera consistente como se observa en la figura 6.4. En dicha figura se aprecian cuatro estados denominados A, B, C y D y dos entradas: X y Y.

La transición de un bloque a otro se realiza a través de líneas denominadas caminos de enlace.

En las cartas ASM, a cada bloque le corresponde una unidad de tiempo y en este lapso se ejecutan todos los bloques de decisión y de salidas condicionales que estén asociados con el mismo estado.

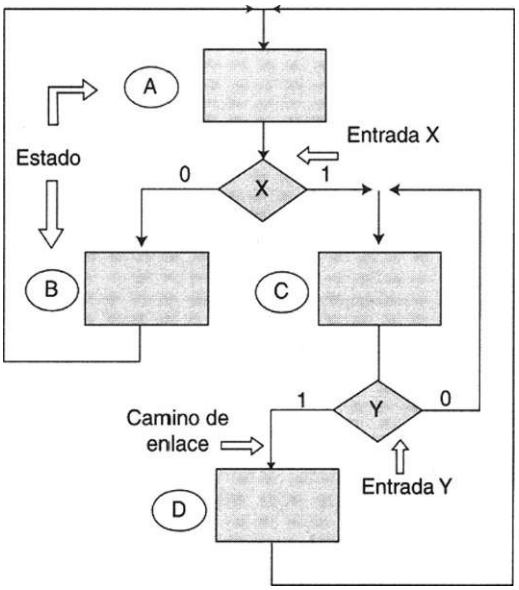


Figura 6.4 Estructura Básica de una carta ASM.

Descripción de una carta ASM

Para describir de manera general los atributos de una carta ASM, consideremos como ejemplo el circuito de la figura 6.5.

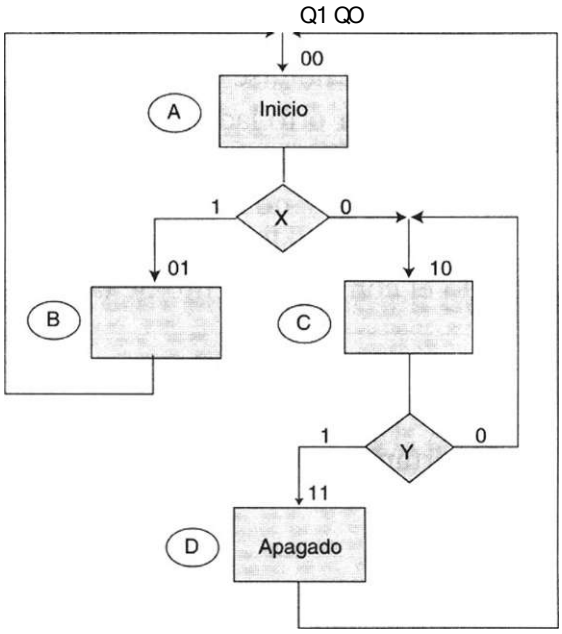


Figura 6.5 Descripción de una carta ASM.

Como puede observarse, la carta ASM está formada por cuatro estados, identificados como A, B, C y D, cada uno con un código binario respectivo; así, el estado A tiene asignado el código 00 y el estado B, el código 01. También puede verse en el estado A que existe una señal de salida denominada Inicio, mientras que en el estado D la señal de salida se define como Apagado. La carta tiene dos señales de entrada identificadas por las variables X y Y.

Esta carta puede describirse mediante la tabla 6.1. Como se puede ver, presenta las columnas Estado presente, Señales de entrada, Estado futuro y Señales de salidas.

	Estado presente			Entradas		Estado Futuro		Salidas
		Q1	QO	X	Y	(Q1, Q0) T + 1		
1	A	0	0	1	*	0	1	Inicio
2		0	0	0	*	1	0	Inicio
3	B	0	1	*	*	0	0	
4	C	1	0	*	0	1	0	
5		1	0	*	1	1	1	
6	D	1	1	*	*	0	0	Apagado

Tabla 6.1 Descripción de una carta ASM.

En la línea 1 (de la tabla 6.1) nos encontramos en el estado A representado por el código Q1, QO = 00, si la señal del sensor X es igual a uno (X = 1) se pasa al estado B código (01)en el tiempo $(Q1, Q0)_{t+p}$ Es importante remarcar que mientras nos encontremos en el estado A la salida activa será *Inicio.* En la línea 2 de nuevo se encuentra el estado A; sin embargo la señal de entrada X adopta el valor de cero X = 0 por lo cual se transfiere al estado C. Debe notarse que la señal de salida aún es Inicio, dado que se trata del estado A; por otro lado la condición de no importa en el sensor de la señal Y (Y = *), se debe a que para pasar del estado A al B o del estado A al C, el valor de entrada Y no es relevante.

El mismo análisis lo puede hacer el lector con relativa facilidad interpretando el comportamiento de la carta ASM de la línea 3 a la 6.

Ejemplo 6.1 Observe la carta ASM de la figura E6.1 y elabore una tabla que describa su comportamiento.

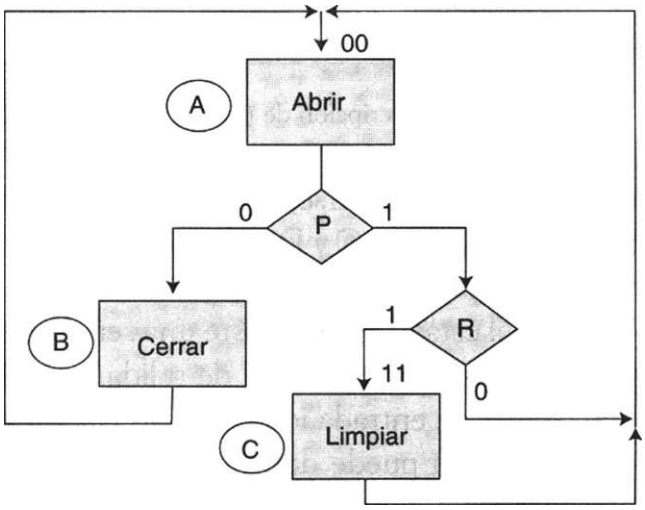


Figura E6.1 Carta ASM.

Solución

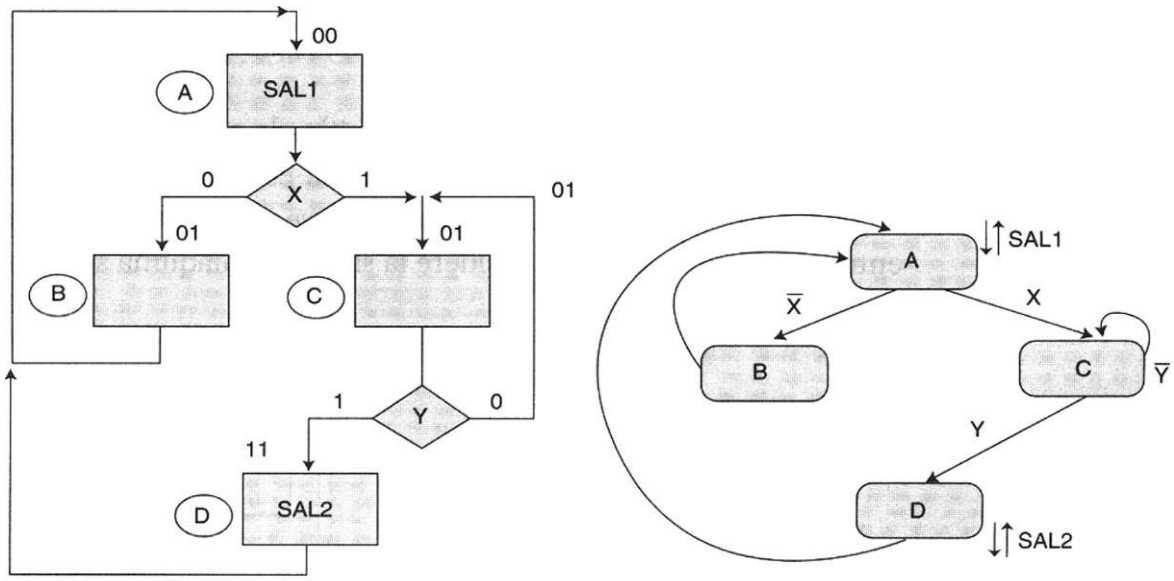
La tabla correspondiente a la carta ASM se muestra a continuación (Tabla E6.1). Observe en la línea 4 cómo es que al estar en el estado B se pasa al estado A sin importar el valor de las señales de entrada P y R.

Estado presente				Entradas		Estado Futuro		Salidas
		Q1	QO	P	R	(Q1, QO) T + 1		
1	A	**0**	**0**	**0**	*	**0**	**1**	Abrir
2		**0**	**0**	**1**	**0**	**0**	**0**	
3		**0**	**0**	**1**	**1**	**1**	**1**	
4	B	**0**	**1**	*	*	**0**	**1**	Cerrar
5	C	**1**	**1**	*	*	**0**	**0**	Limpiar

Tabla E6.1 Descripción de una carta ASM.

6.3 Cartas ASM en comparación con las máquinas de estado

Es relativamente fácil pasar de una carta ASM a un diagrama (máquina) de estados y viceversa por ejemplo, consideremos la carta de la figura 6.6a).



a) b)

Figura 6.6 a) Cartas ASM. **b)** Diagramas de estado.

Como puede observarse, dicha carta está formada por cuatro estados (A, B, C y D), tiene dos señales de entrada (X y Y) y dos salidas (SALl, SAL2). Note cómo estas señales de salida permanecen activas sólo mientras el sistema se encuentra en el estado presente (estructura de Moore).

En la figura 6.6b) se muestra la máquina de estados correspondiente.

Según se aprecia, existen también cuatro estados y la unión entre ellos se presenta a través de sus líneas de entrada (X y Y). La negación de la variable indica la condición cuando las entradas son igual a cero o a uno (Fig. 6.7). Observe también en la figura 6.7b que las señales de salida se denotan mediante la notación (t i).

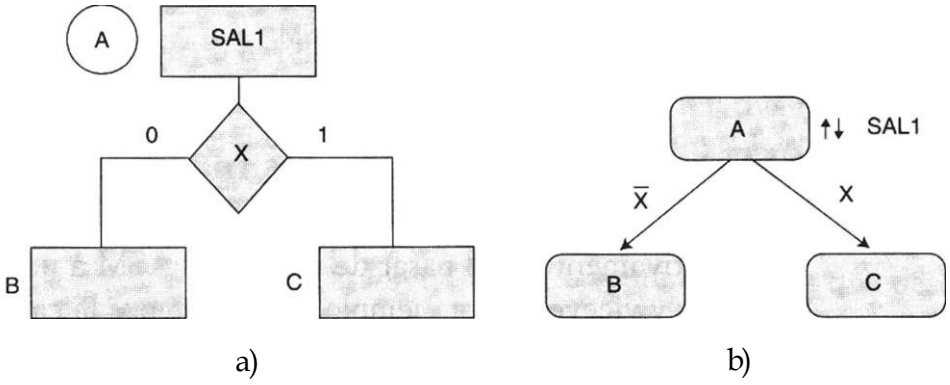


Figura 6.7 a) Negación de variables para indicar la condición de entrada (0 o 1). b) Notación de las señales de salida.

Hasta este momento no se ha especificado el uso de las salidas condicionadas en la carta ASM; éstas se usan en la estructura Mealy cuando la salida depende no sólo del estado en el que se encuentra, sino también de sus entradas. Como ejemplo considere la siguiente máquina secuencial (Fig. 6.8a):

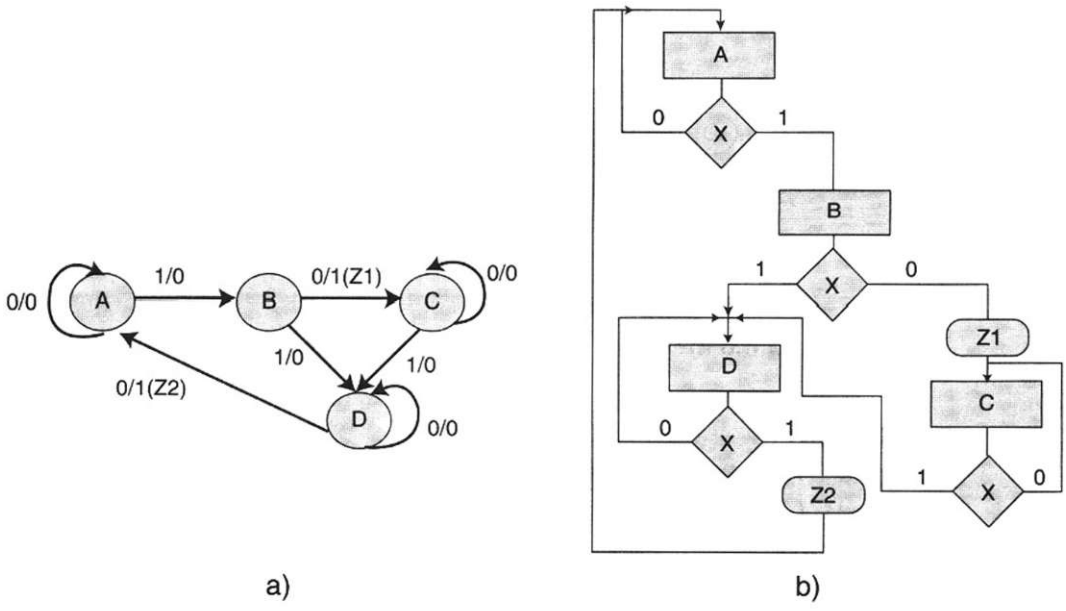
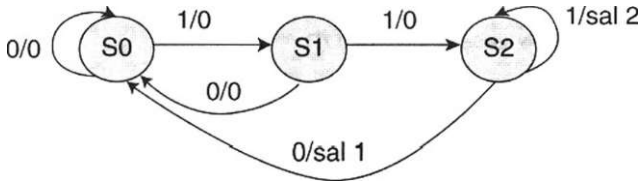


Figura 6.8 a) Estructura de Moore. b) Estructura Mealy en carta ASM.

Como puede observar, cuando la máquina se encuentra en el estado B, si la señal de entrada vale 0 (0/1 (Z1)) se transfiere al estado C y durante este enlace se activa la señal de salida *Z1 = 1.* Igual sucede cuando se halla en el estado D y se dirige hacia el estado A. Observe que la señal de salida que se activa es *Z2 (Z2 = 1).* La carta ASM correspondiente a esta máquina de estado se muestra en la figura 6.8b.

Ejemplo 6.2 Convierta la siguiente máquina de Mealy en una carta ASM.



Solución

La carta ASM correspondiente se muestra en la figura E6.2.

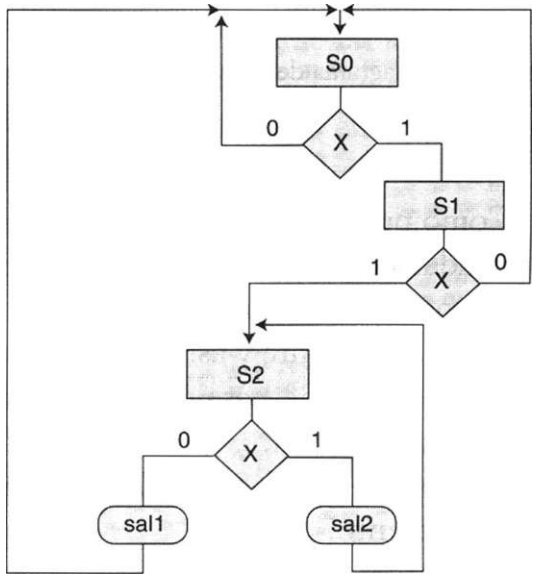


Figura E6.2

6.4 Diseño de controladores mediante cartas ASM

El diseño de cartas ASM tiene mucho que ver con la forma y sentido común de interpretar un problema, para luego visualizar un camino que permita encontrar la solución óptima. Por ejemplo, consideremos la siguiente situación: se nos solicita diseñar un controlador que permita automatizar el funcionamiento de un tren que debe desplazarse de una estación a otra (Fig. 6.9). En cada estación se han colocado sensores que detectan cuando el tren se aproxima al andén y envían al vagón una señal denominada PARO. Al recibir dicha señal, el vagón activa su sistema de frenado y el tren comienza a detenerse en forma automática hasta detenerse y colocarse en los límites de la estación.

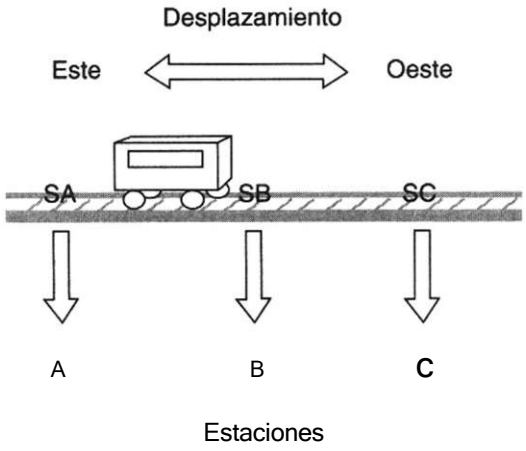


Figura 6.9 Diagrama del controlador de un tren suburbano.

Como puede observarse, en la figura 6.9 se muestran tres estaciones denominadas A, B y C, con sendos sensores cada una (SA, *SB y SC*). Por otro lado, consideremos que cuando el tren se encuentra detenido, abre sus puertas automáticamente y así las conserva durante 15 segundos para permitir la entrada y salida de pasajeros; luego las cierra y continúa su camino hacia la siguiente estación.

Con base en la descripción anterior podemos inferir algunos aspectos de funcionamiento:

1) El tren debe poder moverse de la estación A a la C y viceversa; sin embargo, no se establece con precisión dónde inicia su recorrido. Por lo tanto, si la trayectoria se realiza de A a C y viceversa se produciría un algoritmo con bastantes estados, por lo cual sería conveniente situarse

en la estación B y de ahí desplazarse hacia la dirección ESTE (estación A) o dirección OESTE (estación C).

2) También se interpreta que cuando el tren se aproxima a una estación, el sensor correspondiente envía al vagón una señal de PARO que activa el sistema de frenado y detiene al tren justo en los límites de la estación. La condición anterior establece que el sistema de frenado está predeterminado y sólo basta la señal PARO para iniciar su secuencia de frenado.
3) En la descripción del problema también puede interpretarse que el tiempo de ascenso/descenso de pasajeros se marca mediante un controlador de tiempo independiente al controlador del sistema.

En el diseño de controladores es importante identificar con claridad la parte de control y los subsistemas periféricos que interactúan con él. Desde esta perspectiva es posible ubicar el controlador como una caja negra que permite identificar con claridad el intercambio de señales de entrada y salida que se realizan entre él y sus subsistemas.

En la figura 6.10 se observan las señales de entrada/salida que interactúan en el controlador. La función de estas señales se explica a continuación:

Señales de entrada

- Tiempo Indica el lapso destinado al ascenso o descenso de pasajeros.
- SA Simboliza al sensor colocado en la estación A.
- SB Representa al sensor ubicado en la estación B.
- SC Simboliza al sensor colocado en la estación C.
- DIR Indica hacia dónde se moverá el tren (puede ser hacia el este o el oeste).

Señales de salida

- Este Indica que la dirección del tren será hacia el este.
- Oeste Esta señal indica que la dirección que tomará el tren será el oeste.
- Avance Se envía para que el tren realice su recorrido de una estación a otra.
- P. abierta Señal que permite abrir las puertas del tren cuando se encuentra detenido en una estación.
- P. cerrada Señal que cierra las puertas del tren una vez que el tiempo de ascenso/descenso de pasajeros se ha cumplido.
- Paro Señal que al ser recibida por el tren activa su sistema de frenado para que se detenga lentamente en los límites de una estación.

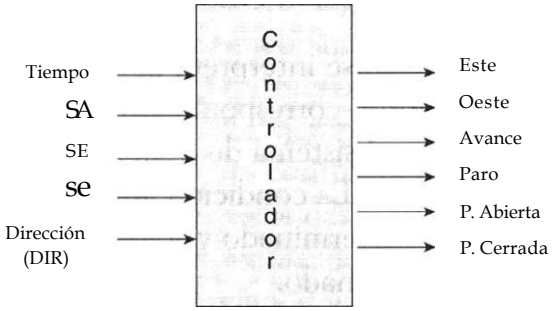


Figura 6.10 Señales de entrada y salida del controlador.

Un posible algoritmo de control "carta ASM" que permite detallar de manera didáctica el comportamiento del sistema es el siguiente (Fig. 6.11):

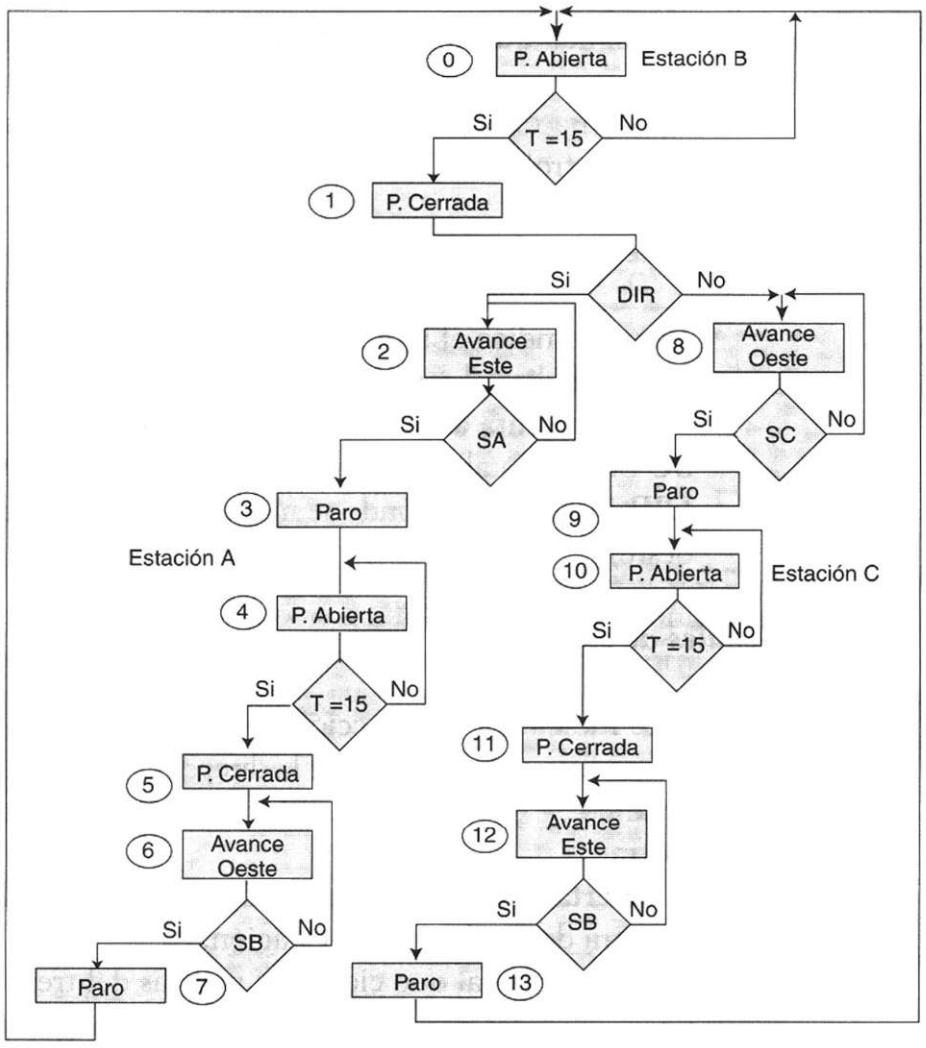


Figura 6.11 Carta ASM de un controlador.

Descripción de algoritmos de control

Estado 0. En un inicio el controlador se encuentra en la estación B y la puerta del vagón se mantiene abierta. Si la entrada tiempo *(T = 15 segundos)* no ha alcanzado su valor final, la puerta continúa abierta *(T – No);* si ya alcanzó el valor de T=15 segundos, la puerta se cierra y el controlador se transfiere al estado 1.

Estado 1. Con la puerta cerrada se elige la dirección a través de la entrada DIR; es decir,

si DIR = Si el vagón se dirige hacia el este (estación A)
si DIR = No el vagón se dirige hacia el oeste (estación B)

Estado 2. En el estado 2 se da la orden para que el tren continúe su movimiento a través de la señal Avance. Como puede observarse, si el sensor SA no detecta la presencia del vagón, la señal de avance permanece inalterable; por el contrario, si el sensor SA detecta el vagón, el controlador (SA = Si) envía una señal de Paro (estado 3), que detiene de forma automática al tren justo en los límites del andén de la estación A.

Estado 4. Con el vagón detenido en la estación A, el controlador envía la señal Puerta abierta, que permite la entrada y salida de pasajeros y mientras el tiempo T = 15 segundos no se cumpla, el tren permanece en la estación A con las puertas abiertas; en caso contrario, las cierra y continúa su avance hacia la estación B.

Se puede deducir con relativa facilidad la descripción general de la carta, en la cual se observaron 14 estados, del estado 0 al estado 13.

En el diseño de algoritmos de control ASM tiene mucho que ver el sentido común y la experiencia del diseñador Por ejemplo, en la carta anterior se puede considerar que los sensores SA, SB y SC actúan de forma similar, por lo cual en lugar de representar tres sensores basta utilizar sólo uno (S), dado que el vagón es incapaz de saber en qué estación se encuentra; en consecuencia, sólo podría tenerse un estado (X) que representa la estación actual. Al efectuar estos arreglos, la carta anterior podría simplificarse de la siguiente manera (Fig. 6.12):

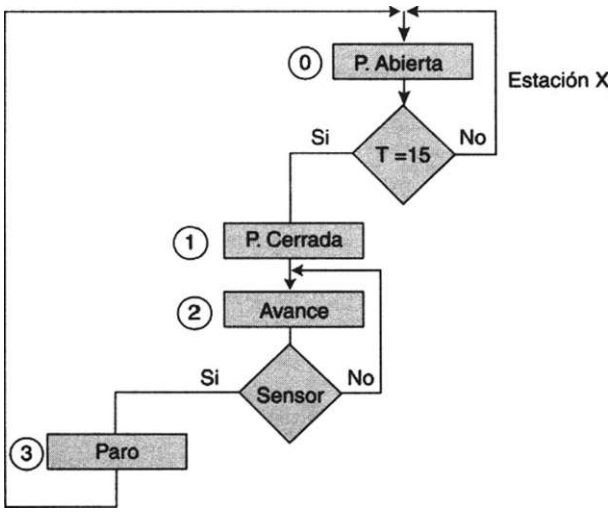


Figura 6.12 Carta ASM simplificada.

Estado 0. El tren está en cualquiera de las estaciones con las puertas abiertas. Si el tiempo T = 15 no se cumple, permanece en la estación actual; si el tiempo ha terminado se da la orden para que la puerta se cierre y luego la orden de Avance para continuar el recorrido (Estado 1 y 2).

Si durante esta trayectoria el sensor detecta la presencia del tren, enviará la señal de Paro, que hará que el vagón se detenga en la estación y abra la puerta (transición del estado 3 al estado 0 en la carta de la Fig. 6.12). Como se advierte, esta carta simplificada sólo utiliza cuatro estados a diferencia de la anterior. La finalidad de ejemplificar estas diferencias es hacer ver que las cartas ASM se construyen con base en la experiencia y conocimiento de los subsistemas que integran una aplicación.

6.5 Diseño de cartas ASM mediante VHDL

Desarrollar un programa en VHDL a través de una carta ASM es relativamente fácil; sin embargo, no es posible recomendar un procedimiento o determinada forma de programar. La experiencia adquirida en los capítulos anteriores ahora adquiere importancia para comprender de manera clara y precisa como se utilizan las palabras reservadas en VHDL.

Para ejemplificar esta situación, veamos el siguiente problema:

I. Descripción

Se requiere diseñar una máquina despachadora de refrescos, la cual está formada por tres módulos (subsistemas) independientes. Cada módulo realiza una función predeterminada, pero hay que diseñar el sistema controlador que gobierne y sincronice cada acción de los subsistemas. A continuación se describe cada subsistema:

a) **Módulo de recolección de monedas.** Recibe las monedas que el cliente introduce en la máquina para obtener el refresco.

Características

- Acepta monedas de $5.00, $10.00 y $20.00.
- Cuenta con un mecanismo que rechaza monedas defectuosas.
- Posee un mecanismo de detección de valor de la moneda; es decir, es capaz de discriminar el valor de la moneda **Menor que precio** (MP) e **Igual a precio** (Precio).

El sistema recibe dos señales de entrada denominadas **Limpieza y Captura.** La primera limpia el sistema y lo deja en condiciones de inicio; la segunda recolecta las monedas que ingresó el cliente.

b) **Módulo de devolución de monedas.** Proporciona el cambio al cliente cuando introduce monedas cuyo monto excede el precio del producto.

Características

- Activa una señal de salida denominada **Cambio,** cuya función es entregar cambio al cliente en monedas de cinco pesos.
- Recibe una señal denominada **Listo cambio** (LC), que indica cuándo se han dado cinco pesos de cambio.

c) **Módulo de servicio.** Su función es entregar el refresco al cliente; sin embargo, el producto sólo se libera cuando la cantidad que proporcionó **el** cliente es igual al valor del refresco.

Características

- Cuenta con una señal de salida denominada **Sirve,** que activa el sensor correspondiente para que el refresco se pueda servir y entregar.

- Posee una señal de entrada denominada Listo servicio (LS), que indica cuándo se entregó el refresco.

d) Controlador. Esta unidad sincroniza las acciones de los diferentes módulos para automatizar el funcionamiento de la máquina despachadora de refrescos.

En la figura 6.13 se observa cada módulo y la señalización entrada/salida que existe entre cada subsistema.

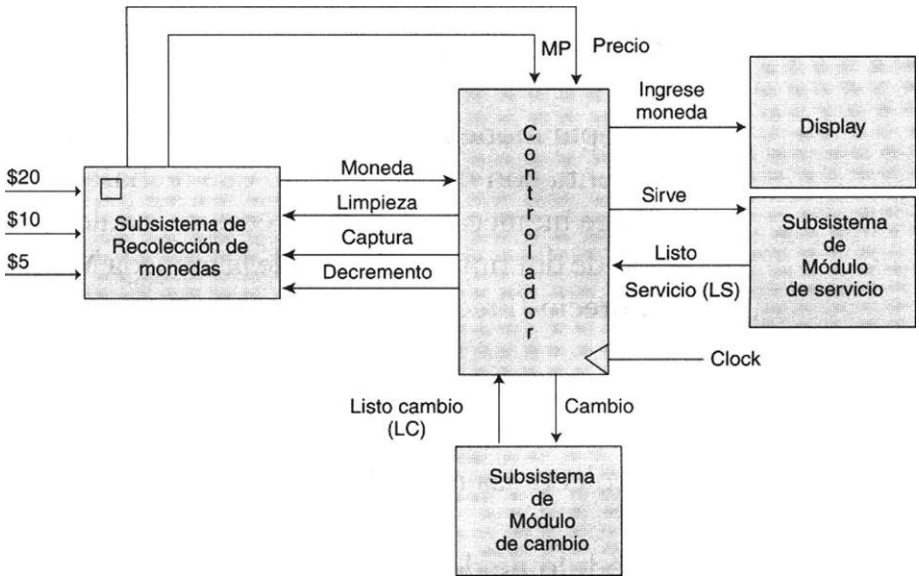


Figura 6.13 Módulos que forman el controlador.

Un posible algoritmo ASM se muestra en la figura 6.14.

Estado A. Se tiene una salida denominada Ingrese monedas, que no es sino una señal que indica al cliente a través de un display que la máquina se encuentra funcionando. Si la señal de entrada Moneda es igual a "No" (M = No), la máquina permanece en el estado A; si la señal Moneda es igual a "Si", la carta conduce al estado B.

Estado B. La señal de entrada Moneda (M) se utiliza para confirmar que el sensor encargado de esta detección ha realizado toda su rutina.

Es importante recordar que existen algunos sensores que suelen encontrarse en el estado de reposo, para luego pasar a su estado de acti' vación y regresar al reposo de nuevo. Este sensor detector de monedas realiza la función anterior.

Estado C. La señal de entrada Menor que precio (MP) se utiliza para indicar al cliente que debe suministrar más monedas; por ejemplo, si (MP = Si), se vuelve al estado A y se solicita al cliente que Ingrese monedas. Si (MP = *No)* se pasa al estado D, donde se pregunta si la cantidad ingresada es igual o mayor que el precio *(Precio).*

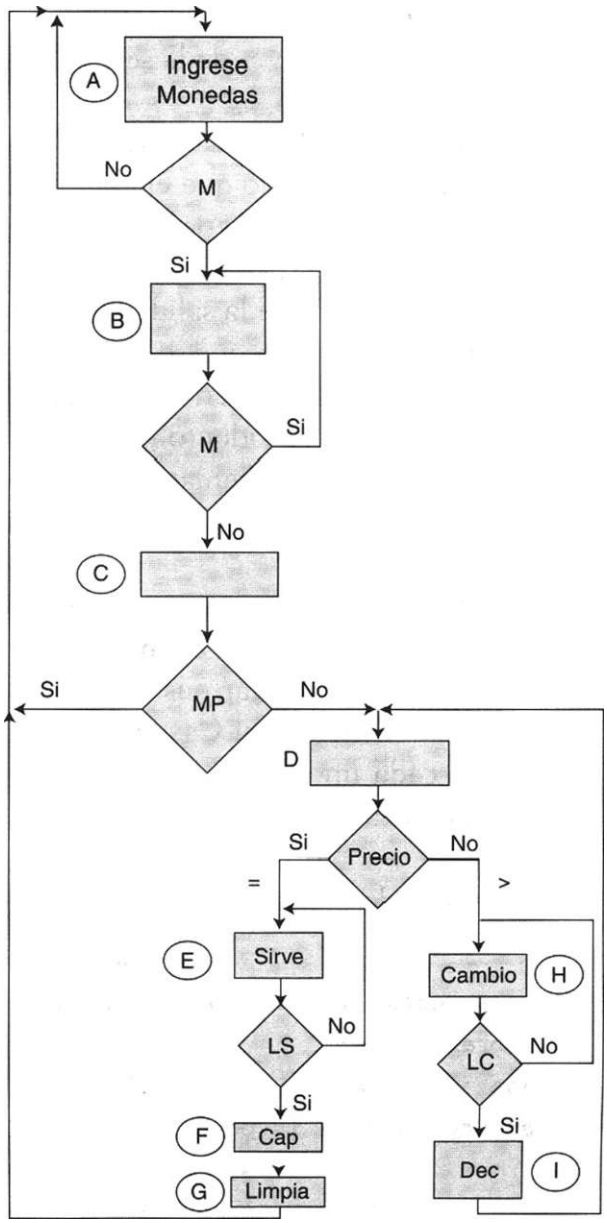


Figura 6.14 Carta ASM del controlador de una máquina de refrescos.

Estado D. En este estado se pregunta si la cantidad ingresada es Igual al precio. Si lo es *(Precio = Si)*, se pasa al estado E donde se activa la salida Sirve, que otorga el servicio. En el estado E la señal Listo servicio *(LS)* se utiliza para indicar al controlador que el sensor ha detectado que ya se entregó el refresco: (LS = *Si),* en caso contrario (LS = No), se repite la petición de servicio. Si éste ya se brindó, la carta se transfiere al estado F para capturar el dinero *(Cap)* y luego al estado G para limpiar el sistema *(Limpia).*

Al volver al estado D se observa que si *Precio = No* "equivale a decir que el precio es mayor al valor del refresco"; en consecuencia, el algoritmo debe devolver cambio a través de la señal Cambio en el estado H.

Estado H. La señal Listo cambio *(LC)* se utiliza para indicar al controlador si el sensor ha detectado que el cambio se dio *(LC = Si)*; en caso contrario (LC = *No),* se repite la petición de cambio.

Estado I. La función de la salida Decrementa *(Dec)* es indicar al módulo de recolección de monedas que se entregaron cinco pesos de cambio, con el objeto de que compare y determine si con el decremento de cinco pesos la cantidad abonada por el producto ya es igual al precio o sigue siendo mayor. Observe cómo el camino de enlace de estado I retroalimenta al estado D en que se pregunta si el pago por el producto es igual o mayor al precio.

Como puede observarse, para describir controladores se requiere conocer a la perfección el funcionamiento de cada uno de sus subsistemas así como la función que realizan sus sensores.

Ahora, la instrucción Case en VHDL puede ayudar a simplificar lo que sucede en cada uno de los estados; de igual manera, el uso de IF o ELSIF puede auxiliar a manejar con simplicidad los rombos de decisión.

En el listado 6.1 se observa la programación del algoritmo anterior.

```
— Programa de la máquina despachadora de refrescos
library ieee;
use ieee.std_logic_1164.all;
entity maquina is port (
        CLK, MONEDA, MP, PRECIO, LC, LS:  in std_logic;
        CAP, LIMPIA, SIRVE, CAMBIO, DEC: out std_logic);
end maquina;
architecture arq_maq of maquina is
   type estados is  (A,B,C,D,E,F,G,H,I);
   signal edo_pres, edo_fut: estados;
     begin
```

Continúa

```
p_estados: process (edo_pres,MONEDA,MP,PRECIO,LC,LS) begin
   case edo_pres is
   when A => CAP<='0'; LIMPIA <= '0'; SIRVE <= '0'; CAMBIO <=
    ' 0 ' ; DEC <= ' 0 ' ;
      if MONEDA -'l' then
         edo_ful <= B;
      else
         edo_fut <= A;
      end if;
   when B => CAP<= ' 0' ; LIMPIA <= '0'; SIRVE <= '0'; CAMBIO < =
    '0'; DEC <= '0';
      if MONEDA ='0' then
         edo_fut <= C;
      else
         edo_fut <= B;
      end if;
   when C =>CAP<=' 0' ; LIMPIA <= '0'; SIRVE <= '0'; CAMBIO < =
    '0'; DEC <= '0';
      if MP ='0' then
         edo_fut <= D;
      else
         edo_fut <= A;
      end if;
   when D => CAP <= '0'; LIMPIA <= '0'; DEC <= '0 *;
      if PRECIO ='0' then
         edo_fut <= H; CAMBIO <= 'l';
      else
         edo_fut <= E; SIRVE <= '1";
      end if;
   when E => LIMPIA <= '0'; CAMBIO <= '0'; DEC <= '0';
       if LS = ' 0[1] then
         edo_fut <= E; SIRVE <= '1';
      else
         edo_fut <= F; CAP <= '1';
      end if;
   when F => LIMPIA <= '0'; SIRVE <= '0'; CAMBIO <= '0'; DEC <= '0';
           edo_fut <= G; CAP <= '1[1]';
   vAien G =>CAP<= ' 0 ' ; SIRVE <= ' 0 ' ; CAMBIO <= ' 0 ' ; LIMPIA <= ' 1' ;
          DEC <= '0';
          edo_fut <= A;
   vAien H => CAP <= '0 ; SIRVE <= '0'; LIMPIA <= 'l'y
          if LC ='0' then
```

Continúa

```
                    edo_fut <= H; CAMBIO <= '1';
                else
                    edo_fut <= I; DEC <= *1' ;
                end if;
            when I => CAP <= '0'; SIRVE <= '0'; LIMPIA <= '1'; CAMBIO
                <= '01;
                    edo_fut <= D; DEC <= '1';
end case;
end process p_estados;
-- continua el programa
-- inicia segundo proceso
p_reloj: process(CLK) begin
     if (CLK'event and CLK='1') then
          edo_pres <= edo_fut;
     end if;
   end process p_reloj;
end arq_maq;
```

Listado 6.1 Descripción de la carta ASM del controlador de la máquina despachadora de refrescos.

De manera general, en el diseño de cartas ASM podemos puntualizar los siguientes aspectos:

1. Para cada combinación válida de variables de entrada, sólo puede existir una salida; esto es, no es posible presentar dos veces la misma situación, debido a que habría dos salidas diferentes.
2. Si es necesario retroalimentar una salida, esto debe hacerse antes del estado y no en éste.
3. Un bloque puede tener varios caminos en paralelo que lleven a la misma salida y más de uno puede estar activo al mismo tiempo. Cuando se tienen caminos en paralelo, se pueden dibujar en forma de serie sin afectar el resultado; sin embargo, un diagrama en paralelo puede hacer la carta más compacta mientras que un diagrama en serie tiene menos probabilidades de llevar a transiciones de estado ambiguas (Fig. 6.15).

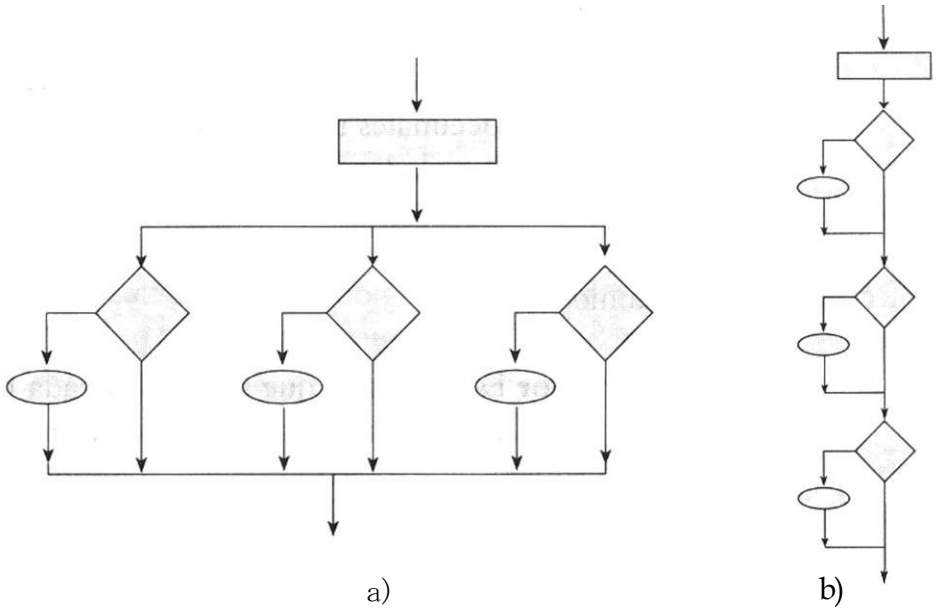


Figura 6.15 a) Estructura en paralelo de una carta ASM. b) Estructura en serie de una carta ASM.

Ejemplo 6.4 Para reafirmar la programación de controladores, consideremos el ejemplo mostrado en la figura E6.4.

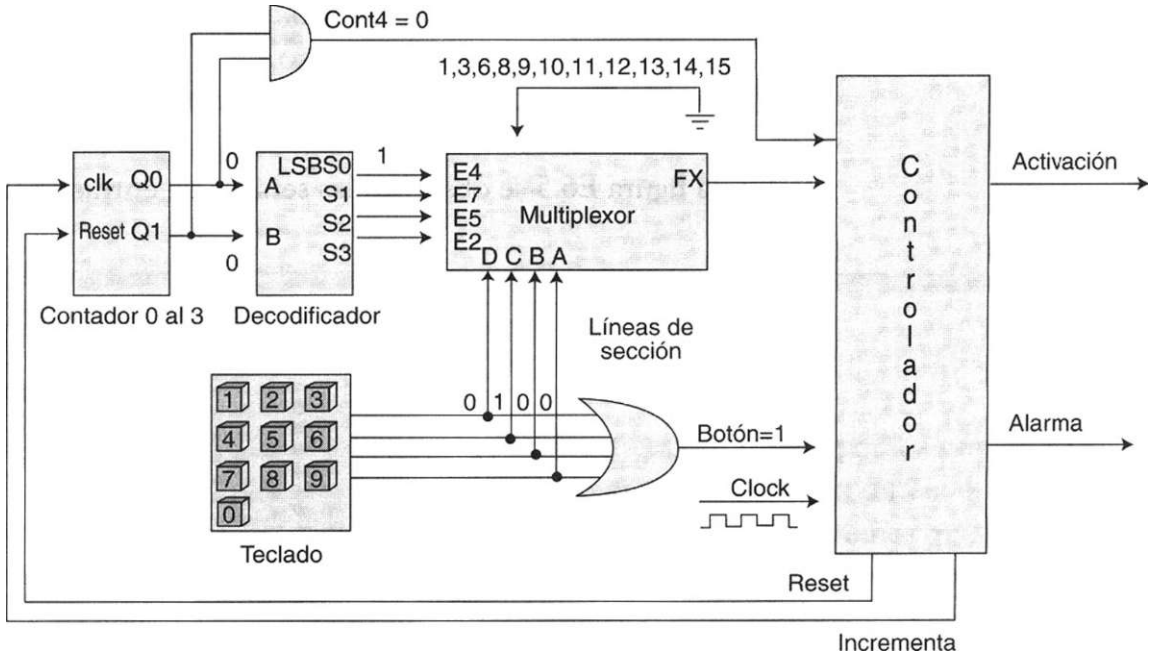


Figura E6.4 Diagrama a bloques del sistema de seguridad.

Como puede observarse, en el diagrama se presenta cada uno de los módulos que integran un sistema electrónico. **Este sistema está diseñado para abrir una caja de seguridad basada en una clave de acceso:** si la clave de cuatro números decimales introducidos en secuencia a través del teclado es la correcta, el controlador origina una señal de salida denominada **Activa* ción,** encargada de abrir la caja; en caso contrario, activará una señal de salida denominada **Alarma.** Con base en lo anterior, la descripción detallada de cada módulo sería:

I. Controlador Es la unidad que sincroniza cada una de las acciones de los módulos para automatizar el sistema de seguridad. A continuación se exponen las funciones de sus señales de entrada/salida.

- **Salida Reset.** Inicializa el contador y lo coloca en el estado de conteo cero.
- **Salida Incrementa.** Señal utilizada como reloj para incrementar en uno el contador del 0 al 3.
- **Salida Activación.** Activa la apertura de la caja de seguridad.
- **Salida Alarma.** Indica que la clave introducida fue incorrecta.
- **Entrada FX.** Indica si el número decimal introducido forma parte de la clave correcta.

Si *FX = 1* el número introducido es correcto, si *FX* = 0, se trata de un número incorrecto.

- **Entrada Cont4.** Señal de aviso que indica a través de Cont4 = **1** que se ha tecleado el cuarto número.
- **Entrada Botón.** Indica que se ha presionado una tecla.

En la figura E6.5 se observan las señales de entrada/salida del controlador.

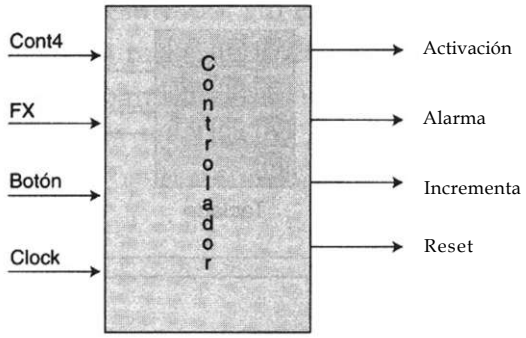


Figura E6.5 Descripción de las señales de entrada/salida del controlador de la caja de seguridad.

La relación de los módulos mostrados en la figura E6.4 y el controlador se describe por medio de la carta ASM correspondiente (Fig. E6.6).

Clave correcta

Estado A. En un inicio el controlador envía la señal de salida Reset, con el objeto de colocar al contador en el estado cero (Q1=0 ,Q0=0). Con esta condición de salida del contador, el decodificador coloca en estado alto la salida SO (SO = 1), al mismo tiempo la salida *Cont4* adopta el valor de cero *(Cont4* = Q1 QO).

Estado B. En este estado nuestro controlador está en espera de que se presione la tecla correspondiente al primer número de la secuencia de la clave correcta.

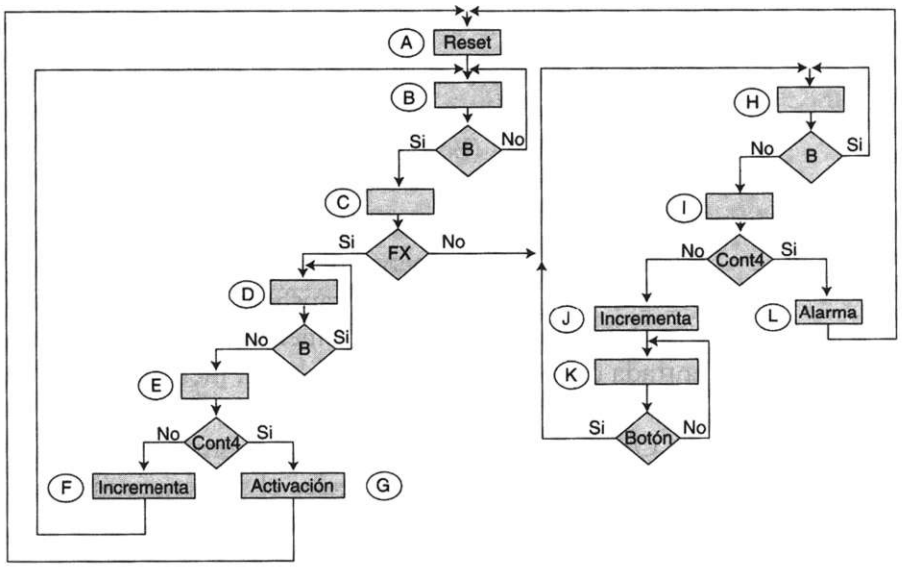


Figura E6.6 Carta ASM del sistema de seguridad.

Estado C. Supongamos que la clave correcta está formada por los números (4, 7, 5 y 2), de suerte que al principio el primer dígito tecleado será el 4. Cuando esto ocurre, la señal de botón toma el valor de uno. Observemos que la entrada a la compuerta OR está definida por el número binario equivalente al dígito decimal 4 (0100): (Figura E6.7). Note que este valor binario permite a su vez seleccionar la entrada E4 del multiplexor, la salida FX toma el valor de uno (FX =1), haciendo que el controlador se transfiera al estado D. Es importante observar la figura E6.7, en la cual se puede apreciar que inmediatamente

después de presionar el número 4, se selecciona la entrada cuatro del multiplexor, entrada conectada a la salida SO del decodificador, haciendo que la función FX sea igual a uno.

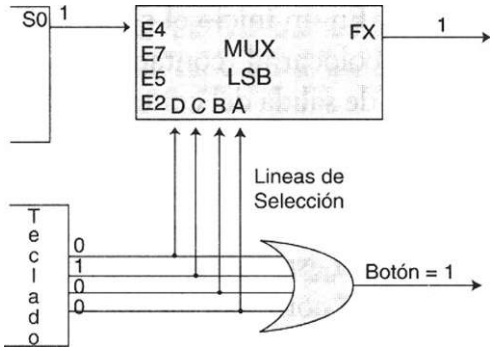


Figura E6.7 Funcionamiento de la señal FX del multiplexor.

Estado D. En este estado el controlador pregunta si el usuario aún tiene presionada la tecla correspondiente al número 4 (Botón = Si) o si la tecla ya fue soltada (Botón =No).

Estado E. Se evalúa el valor de la señal de entrada Cont4, la cual, como se recordará, en este momento tiene el valor de cero *Cont4 = No.* Lo anterior ocasiona que el controlador transfiera su secuencia al estado F, donde se activa la señal de salida de incremento. Cuya función es enviar un pulso a la entrada de reloj del circuito contador, incrementando su cuenta en uno.

Observe en la carta ASM que el controlador va del estado F al estado B en espera de que el usuario teclee el siguiente número de la clave correcta y luego repetir la secuencia anterior.

Estado F. Cuando el contador llega al conteo de tres *(Ql = l y* Q0 = *1),* la señal de entrada *Cont4* adopta el valor de uno *(Cont4= Si),* con lo que el controlador pasa al estado G y se genera la señal de salida Activación. Dicha señal abre la puerta del sistema de seguridad y hace que el sistema se transfiera al estado inicial (A), en espera de que el usuario introduzca de nuevo la clave de entrada.

Clave incorrecta

Ahora consideremos el caso en que se introduce un número incorrecto como clave de entrada. Por ejemplo, imaginemos que nos encontramos en el estado A con las siguientes condiciones: el contador se halla en el conteo de cero,

la señal *Cont4* es igual a cero y la salida activa del decodificador SO igual a uno (SO = 1). En ese momento (estado B) presionamos una tecla equivocada, supongamos que es el número seis "6". Como puede observarse, la entrada E6 del multiplexor está conectada a tierra; es decir, E6 = 0 (Fig. E6.4); en consecuencia, FX = 0. Según se advierte, con este valor de FX la carta ASM se dirige hacia el estado H.

Estado H. Al igual que en el caso del estado C, el controlador solicita que se le confirme si el botón ya no está presionado; si es así, pasa al estado 1.

Estado I. Aquí se evalúa la señal de entrada Cont4, la cual —como ya se mencionó— tiene un valor de cero, Cont4 — No. Esto ocasiona que la carta ASM se dirija al estado J y active la señal de incrementa, la cual sirve para aumentar en una unidad el contador externo.

Estado K. En este momento el controlador está en espera de que se ingrese otro número. Observe que sin importar la cifra que se presione (Botón = Si), la carta se transfiere al estado H "retroalimentación necesaria para que el controlador no se salga de ese ciclo y sólo espere el conteo de Cont4 = Si para activar la alarma del sistema".

En el listado 6.2 se observa el código VHDL correspondiente a este ejemplo.

```
library ieee;
use ieee.std_logic_1164.all ;
entity c_alarma is port(
 elk,fx,cont4,boton: in std_logic;
 reset,ine,activa,alarma: out std_logic);
end c_alarma;
architecture arq_alarma of c_alarma is
    type estadosl is (A,B,C,D,E,F,G,H,I,J,K,L);
    signal edo_pres, edo_fut: estadosl;
      begin
         p_estadosl: process (edo_pres, elk, fx, cont4, boton) begin
            case edo_pres is
           when A => reset <= '0';inc <= '0';activa <= '0';alarma <= '0';
                edo_fut <= B;
           vàien B => reset <= '0';inc <= '0';activa <= '0';alarma <= "0' ;
      if boton = ' 1 ' then
       edo_fut <= C;
      else
       edo_fut <= B;
      end if;
     when C => reset <= '0'; ine <= '0';activa <= '0';alarma <= '0[1] ;
```

Continua

```
    if fx = '1' then
     edo_fut <= D;
    else
     edo_fut <= H;
    end if;
   when D => reset <= '0';inc <= '0';activa <= '0';alarma <= '0';
     if boton = '1' then
      edo_fut <= D;
     else
      edo_fut <= E;
     end if;

  when E => reset <= '0';inc <= '0';activa <= '0';alarma <= '0';
     if cont4 = '1' then
       edo_fut <= G;
     else
      edo_fut <= F;
     end if;
    when F => reset <= '0';activa <= '0';alarma <= '0'; inc <= '1';
       edo_fut <= B;
    when G => reset <= '0';activa <= '1';inc <= '0'; alarma <= *0';
       edo_fut <= A;
    when H => reset <= '0';inc <= '0';activa <= '0';alarma <= '0';
     if boton = '1' then
      edo_fut <= H;
     else
      edo_fut <= I;
     end if;
    when I => reset <= '0';inc <= '0'; activa <= '0';alarma <= '0';
      if cont4 = '1' then
      edo_fut <= L;
     else
      edo_fut <= J;
     end if;
    when J => reset <= '0';inc <= 'l';activa <= '0';alarma <= '0';
      edo_fut <= K;
    when K => reset <= '0';inc <= '0';activa <= '0';alarma <= '0';
      if boton = '1' then
      edo_fut <= H;
     else
      edo_fut <= K;
     end if;
    when L => reset <= '0';inc <= '0'; activa <= '0' ; alarma <= '1';
      edo_fut <= A;
   end case;
end process;
```

Continúa

```
reloj: process (elk) begin
       if (elk'event and elk = '1' )then
         edo_pres <= edo_fut;
       end if;
  end process;
end architecture arq_alarma;
```

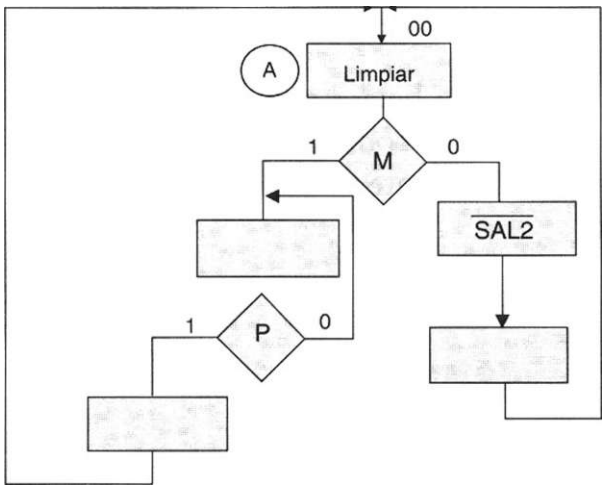
Listado 6.2 Descripción del controlador del sistema de seguridad.

Como hemos podido notar, el diseño de cartas ASM depende en gran medida del conocimiento que el diseñador tenga de cada uno de los subsistemas que integran una aplicación.

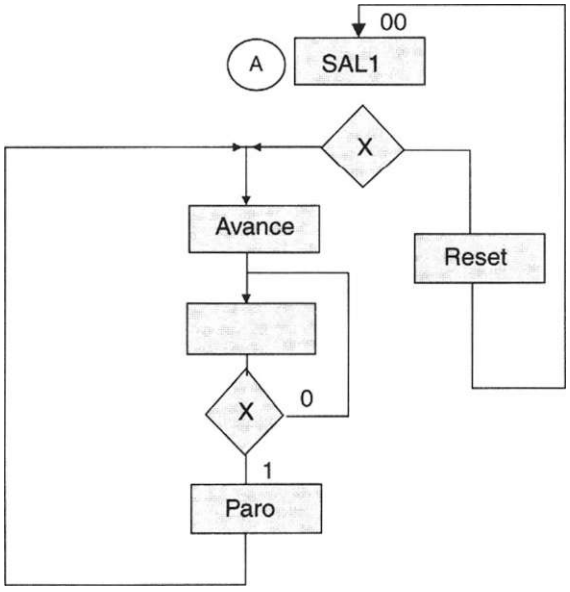
Ejercicios

Estructura de una Carta ASM

6.1 Realice una tabla y describa el comportamiento de las siguientes cartas ASM.



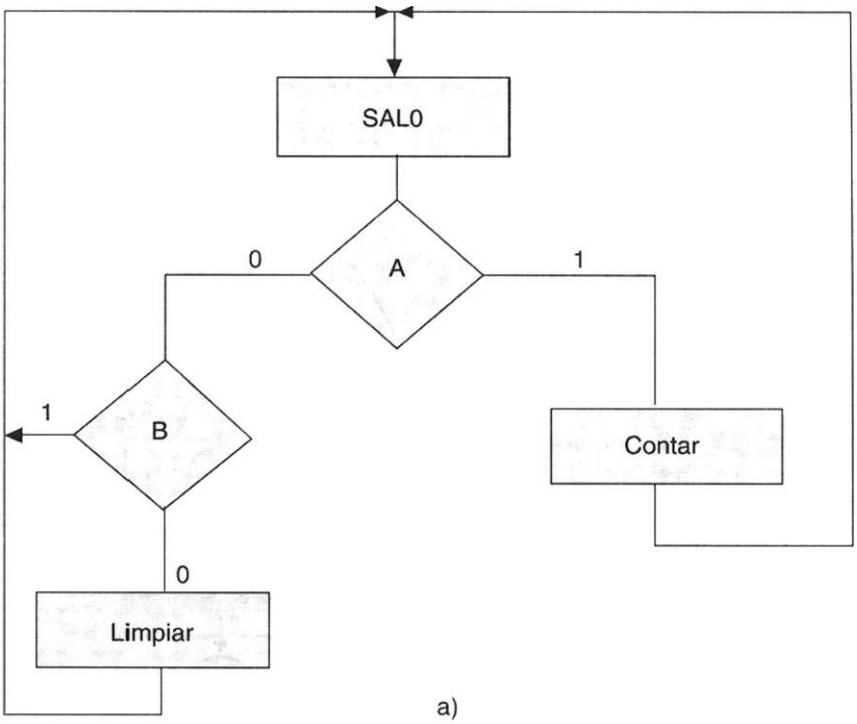
a)



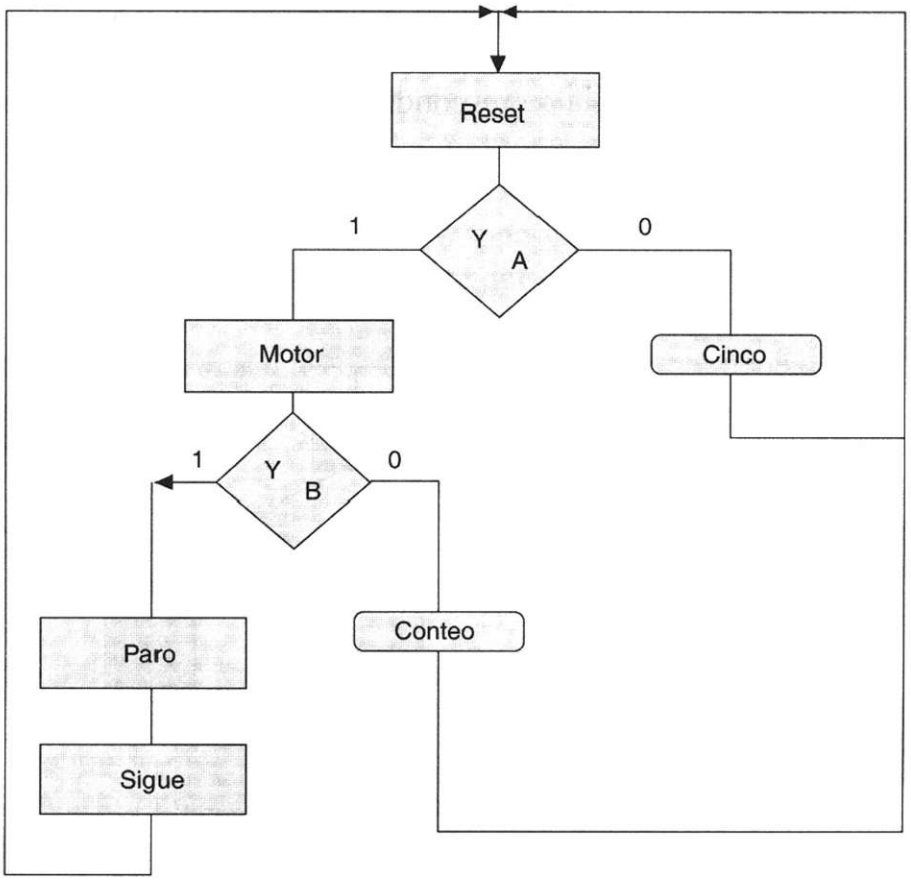
b)

Cartas ASM *contra máquinas de estado*

6.2 Obtenga los diagramas de estado para cada una de las siguientes cartas ASM.

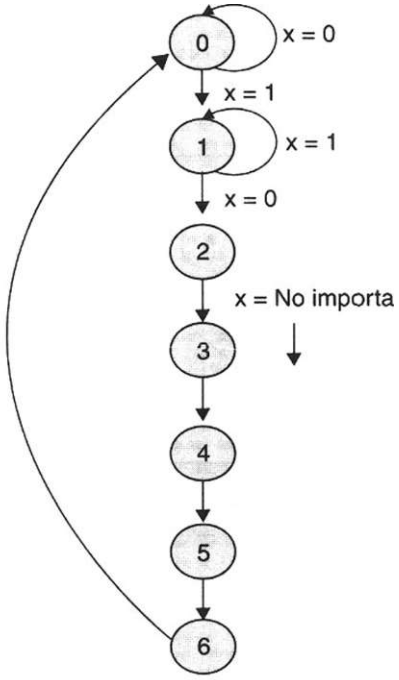

SAL0
A
0
1
B
1
0
Contar
Limpiar

a)


Reset
Y
A
1
0
Motor
Cinco
Y
B
1
0
Paro
Conteo
Sigue

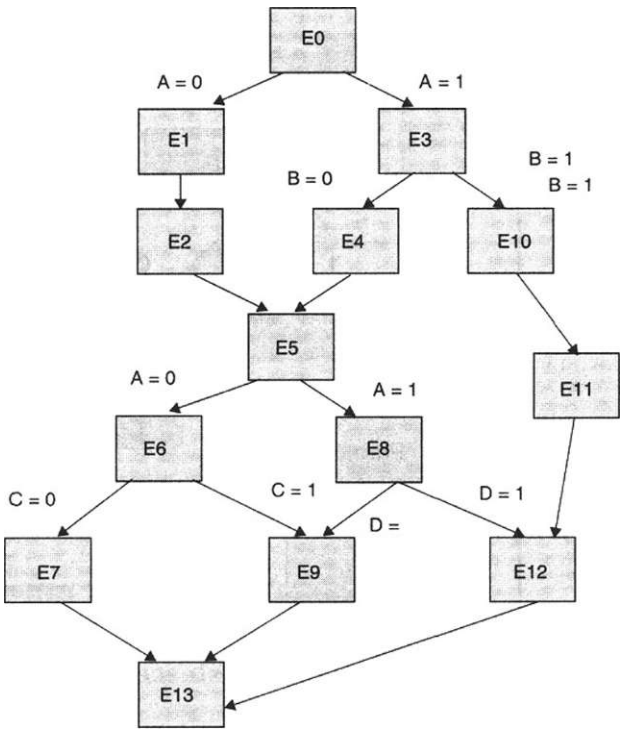
b)

6.3 Dibuje la correspondiente carta ASM del siguiente diagrama de estados. Figura 6.3

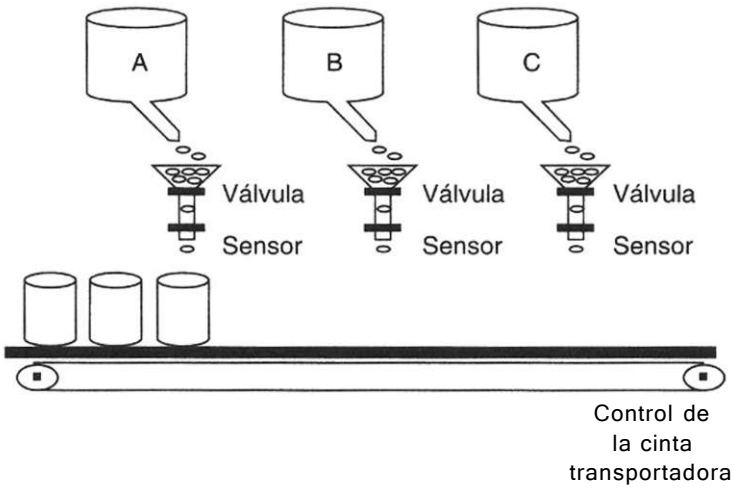


Diseño de controladores mediante cartas ASM

6.4 Dibuje la correspondiente carta ASM del siguiente diagrama de estados.



6.5 Se requiere llenar un frasco con 3 diferentes tipos de pastillas. Cada una de estas pastillas está almacenada en tres recipientes diferentes denominados A, B y C; el llenado debe realizarse de forma secuencial, es decir primero las pastillas del recipiente A, posteriormente el B y finalmente el C, tal y como se observa en la siguiente figura.



En cada frasco deben introducirse 10 pastillas de cada tipo por lo que el alimentador de pastillas de cada recipiente debe detenerse cuando el sensor correspondiente haya detectado que el número de pastillas es igual a 10, en ese momento el motor es encendido y hace que la banda transportadora desplace el frasco hasta el siguiente alimentador de pastillas, el cual repite el proceso anterior y transfiere al frasco hacia el tercer alimentador, el cual al terminar su conteo de pastillas correspondiente activa la banda y hace que el frasco se transfiera hacia otra parte del proceso.

a) Identifique variables de entrada y salida del controlador
b) Diseñe la carta ASM que automatice el proceso

6.6 Diseñe el controlador de una contestadora telefónica cuyo funcionamiento se describe a continuación.

Inicialmente el teléfono se encuentra activado por el usuario para que sea contestado de forma manual o a través de la contestadora (Este estado de activación se muestra con el encendido de un LED incorporado al aparato) El usuario tiene la opción de contestar el teléfono en el primero, segundo o tercer ring , la contestadora deberá activarse si el usuario no descuelga la bocina al tercer ring y activa la grabación cuyo mensaje es:

"Muy buenos días, integración de sistemas electrónicos a sus órdenes, en este momento no podemos atenderle. Deje su mensaje después de la señal. bip,bip,bip...En este momento la persona que llama deja el mensaje (durante el tiempo que dura el mensaje se ilumina en la contestadora un display que dice grabando) al termino del mensaje se desactiva la contestadora en su modo de grabación y vuelve a su estado inicial.

a) Identifique variables de entrada y salida del controlador
b) Diseñe la carta ASM que automatice el proceso

6.7 Diseñe un controlador digital que permita controlar las luces traseras de un automovil .Hay tres luces en cada lado y operan en secuencia para mostrar la dirección en que se va a dar vuelta .La secuencia que se va a encender depende de la intención del conductor para dar vuelta a la izquierda o a la derecha a continuación se muestra la secuencia correspondiente.

Vuelta izquieda			Vuelta derecha		
L3	L2	L1	R3	R2	R1
0	0	0	0	0	0
0	0	1	1	0	0
0	1	1	1	1	0
1	1	1	1	1	1

a) Identifique variables de entrada y salida del controlador
b) Diseñe la carta ASM que automatice el proceso

6.8 Se requiere diseñar el controlador de un ratón electrónico equipado para grabar de las 12:00 pm a las 5:00am,si existen personas dentro de las oficinas de la compañía ACSA CORPORATION S.A. DE C.V. El ratón en su modo de encendido siempre se encuentra grabando, la forma en la que el ratón se desplaza dentro de las instalaciones de tan importante compañía es la siguiente:

El ratón debe maniobrar girando cuando entre en contacto con un obstáculo, la nariz del ratón tiene un sensor cuya salida es X= 1 en caso contrario, siempre que se encuentre en contacto con un obstáculo e s X = 0.

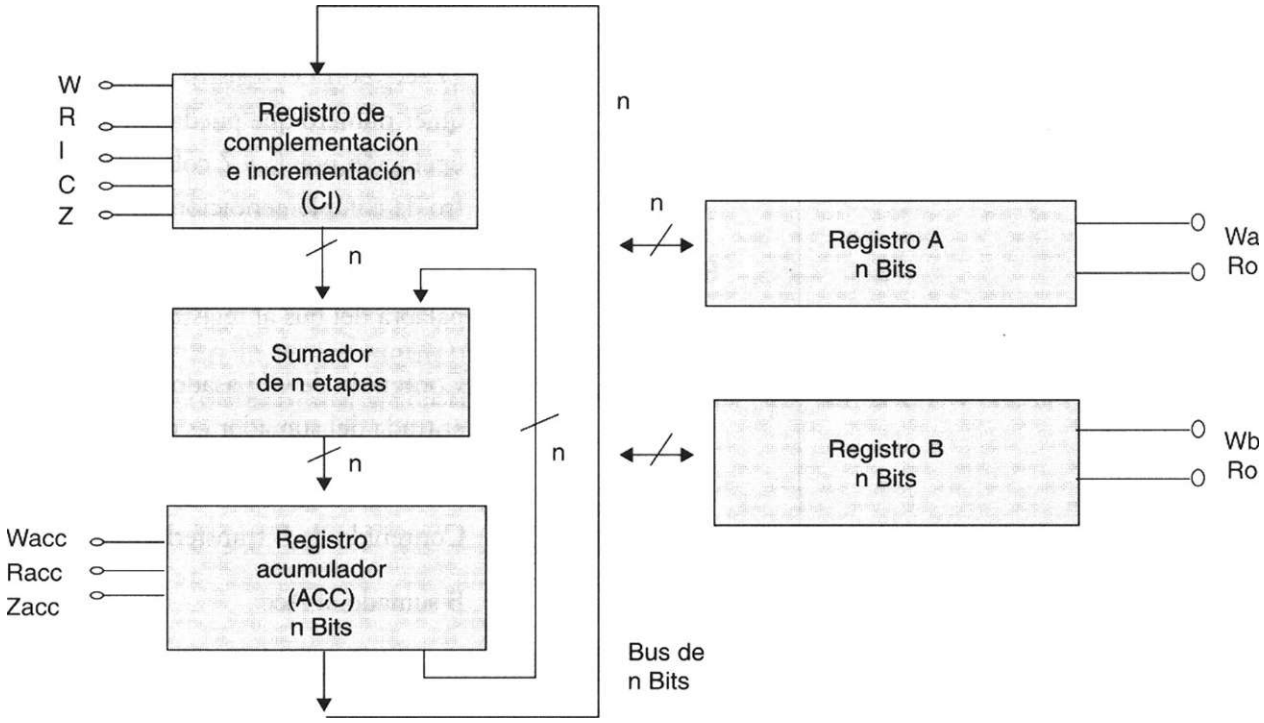
El ratón tiene algunas líneas de control, entre ellas Zl = l,que gira al ratón hacia la izquierda, Z2= 1 que gira el ratón hacia la derecha.

Cuando el ratón encuentra un obstáculo deberá girar hacia la derecha hasta no detectar obstáculo alguno, la siguiente vez que detecte algo similar el ratón deberá girar hacia la izquierda hasta que no haya obstáculo y así sucesivamente.

Cuando el tiempo de grabación ha transcurrido, el ratón enviará una señal inalámbrica al puesto de control y permanecerá en estado de alto hasta el momento en el cual el inspector de seguridad lo apague y lo recoja.

a) Identifique variables de entrada y salida del controlador
b) Diseñe la carta ASM que automatice el proceso

6.9 Se requiere diseñar un sistema de control para el circuito mostrado en la siguiente figura.



El sistema debe calcular el valor de la suma o diferencia aritmética de dos números binarios de n bits. Específicamente queremos calcular las sumas y diferencias a + b,

a - b, -a + b y -a - b. El mecanismo por el que estos números se introducen en los registros a y b no se muestra en la figura.

Todos los registros y el sumador acomodan n bits. El registro de complementación e incrementación (CI) se conecta al sumador, el sumador al acumulador y el acumulador se vuelve a conectar al sumador, todas las

conexiones se realizan con n líneas. Ya que estas conexiones de n bits están dedicadas, es decir cada una sirve para una sola función de transmisión (no son buses).

Existe un bus de n bits al y del cual podemos transferir los contenidos de los registros A y B . Esta transferencia, factible de dos formas, se indica por una flecha bidireccional. A todos los registros se aplica una señal de reloj común, que no está indicada. Cuando Wa = 1, en el flanco de disparo de reloj se transfiere una palabra del bus al registro A ; es decir, se escribe una palabra en el registro. Cuando Ra = 1, se lee una palabra del registro al bus. Algo similar se aplica a la respuesta del registro B mediante las señales Wb y Rb .Cuando Wacc — 1, la salida del sumador se registrará en el acumulador y cuando Racc = 1, el contenido del acumulador se colocará en el bus.

La descripción de la secuencia que debe de seguirse se muestra en la siguiente tabla.

Ciclo de reloj	Líneas de control a poner en 1 lógico	Comentario
1	Zacc,	ZZacc ,Borra el registro acumulador, de cualquier número que pueda haber quedado en una operación previa y Z coloca las salidas del registro de complementación en cero.
2	Ra,W	Lee el data del registro A en el bus y escribe la palabra del bus al registro CI
3	R.Wacc	Contenido de CI pasado por el sumador (la otra entrada del sumador es cero) y registrado en Acc.
4	Rb,W	Contenido de B transferido a CI
5	R, Wacc	B sumado al Acc
6	Racc,Wa	Contenido del Acc transferido al registro

Tabla de funcionamiento

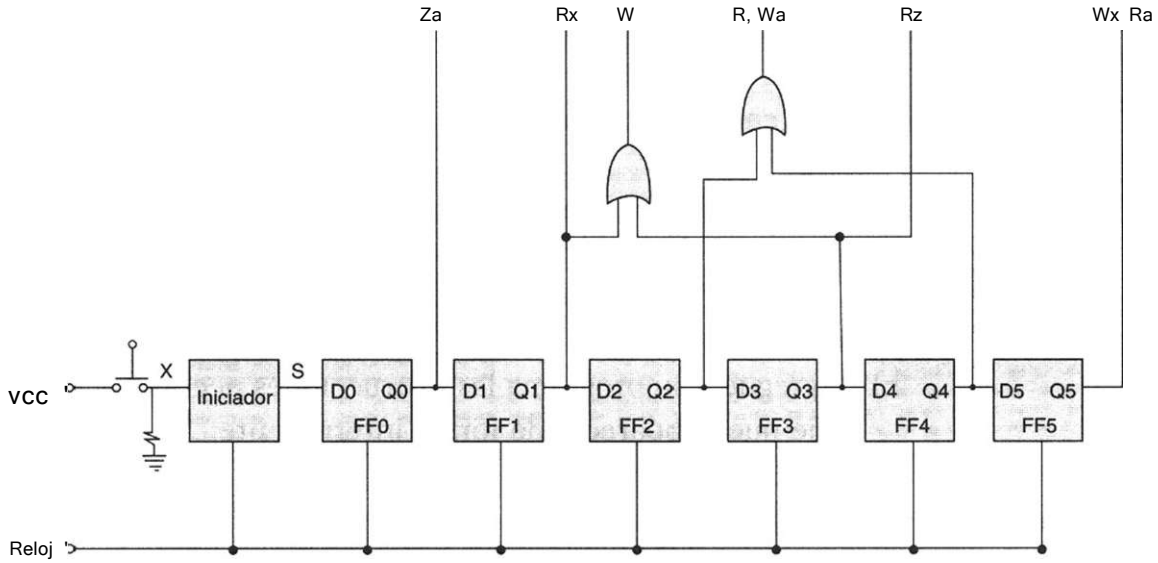
a) Identifique variables de entrada y salida del controlador
b) Diseñe la carta ASM que automatice el proceso

Diseño de cartas ASM mediante VHDL

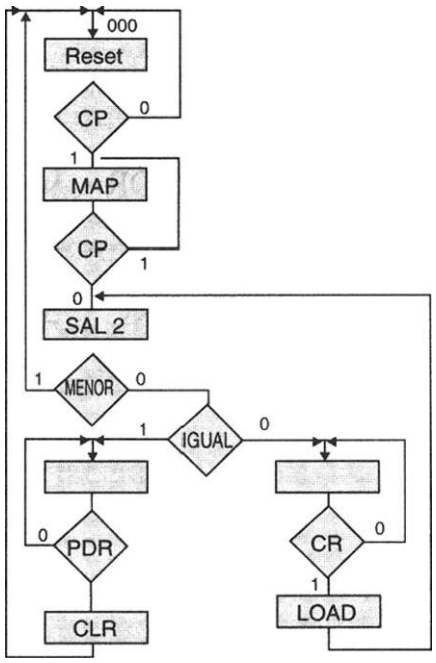
6.10 En la figura siguiente se muestra el diagrama de un controlador de anillo, diseñado para activar en orden secuencial las señales de salida Za,

Rx, W, R, Wa, Rz, Wx, Ra mediante el valor de 1 lógico. El valor de inicio se obtiene mediante la señal de entrada X , este valor debe irse desplazando en cada pulso de reloj.

a) Obtenga el diagrama de estados del controlador
b) Realice la programación correspondiente en VHDL



6.11 En la figura siguiente se muestra la carta ASM del controlador de una remachadora electrónica. Obtenga el programa en VHDL para dicha carta.



6.12 Realice un programa para la carta del ejercicio 6.5

6.13 Realice un programa para la carta del ejercicio 6.6

6.14 Realice un programa para la carta del ejercicio 6.7

6.15 Realice un programa para la carta del ejercicio 6.8

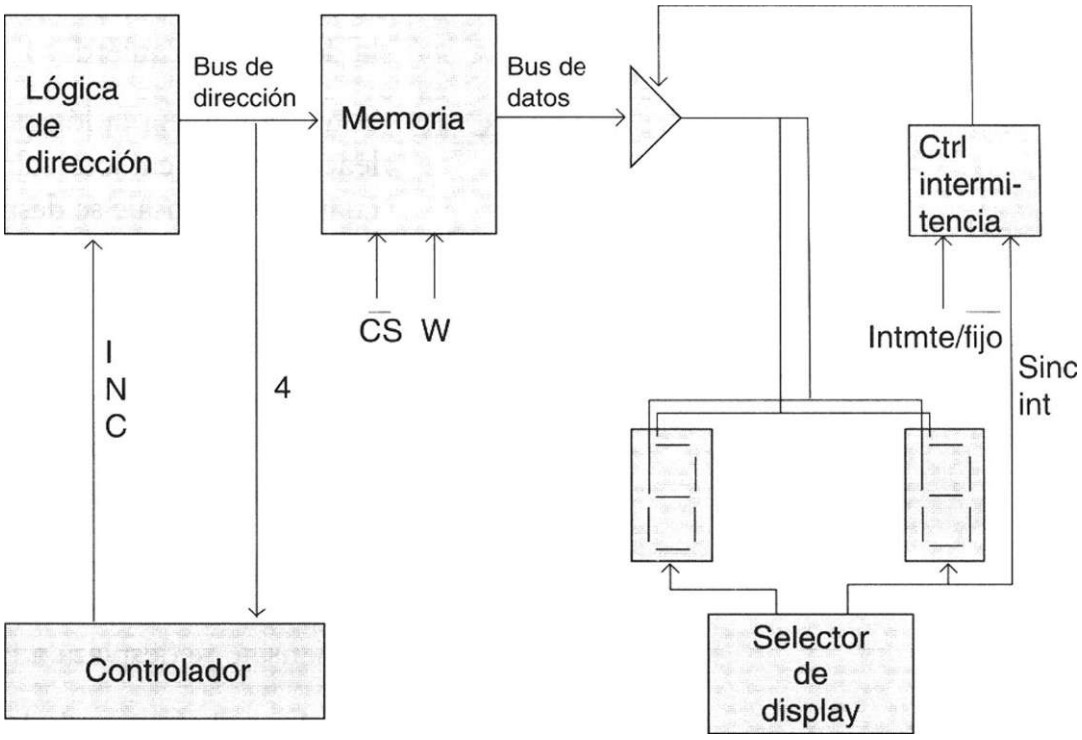
6.16 Realice un programa para la carta del ejercicio 6.9

6.17. Para el circuito mostrado a continuación.

a) Realice el programa del circuito controlador de tal manera que se utilicen 5 displays para visualizar un mensaje alfanumérico, cuyos códigos de caracter se encuentran almacenados en el bloque denominado como memoria.
b) Agregue a su programa las instrucciones necesarias para que el mensaje pueda aparecer de forma intermitente.

A contunuación se describen los elementos y señales de control:

LÓGICA DE DIRECCIÓN	Proporciona la dirección del caracter que se va a visualizar en el display correspondiente.
MEMORIA	Contiene los caracteres visualizar en los displays
SELECTOR DEL DISPLAY	Habilita el display en el que se visualiza el caracter
CONTROLADOR	Circuito que sincroniza las operaciones de la lógica de dirección
SINC	Señal que reinicia el barrido de la memoria.



Diseño de cartas ASM *mediante VHDL*

6.18 Implemente un circuito que utilice 5 displays para visualizar un mensaje cuyos códigos de caracter se encuentran almacenados en memoria. El controlador deberá ser tal que el mensaje se desplace a la izquierda o a la derecha. A continuación se describen los elementos y señales de control:

LOGICA DE DIRECCIÓN DE DESPLAZAMIENTO — Determina la dirección del primer caracter a ser leído dependiendo si el desplazamiento es hacia la izquierda o hacia la derecha.

LOGICA DE DIRECCIÓN — Proporciona la dirección del caracter que se va a mostrar en el display correspondiente.

MEMORIA — Contiene los caracteres a mostrar en los display.

SELECTOR DE DISPLAY — Habilita el display en el cual se muestra el caracter.

CONTROLADOR — Circuito que sincroniza la operación de las lógicas de dirección de desplazamiento y de dirección.

LD1 Señal que habilita la carga de la dirección donde se encuentra el último caracter a desplazar hacia la derecha o izquierda.

CLR1 Señal que inicia la lógica de dirección para que lea el primer caracter almacenado en memoria cuando el mensaje se desplaza hacia la izquierda.

IZQ/DER Señal que controla el desplazamiento hacia la izquierda o derecha del mensaje.

UP Señal que incrementa la dirección que contiene la lógica de dirección de desplazamiento cuando el mensaje se desplaza a la izquierda.

DWN Señal que decrementa la dirección que contiene la lógica de dirección de desplazamiento cuando el mensaje se desplaza a la derecha.

LD2 Señal que habilita a la lógica de dirección para cargar la dirección que proporciona la lógica de dirección de desplazamiento.

CLR2 Señal que permite se reinicie el barrido de memoria al leer el último caracter del mensaje.

CS Señal activa en bajo, que habilita la memoria para una operación de lectura o escritura.

W Señal que controla la lectura o escritura de memoria.

W = 0, escritura
W = 1, lectura

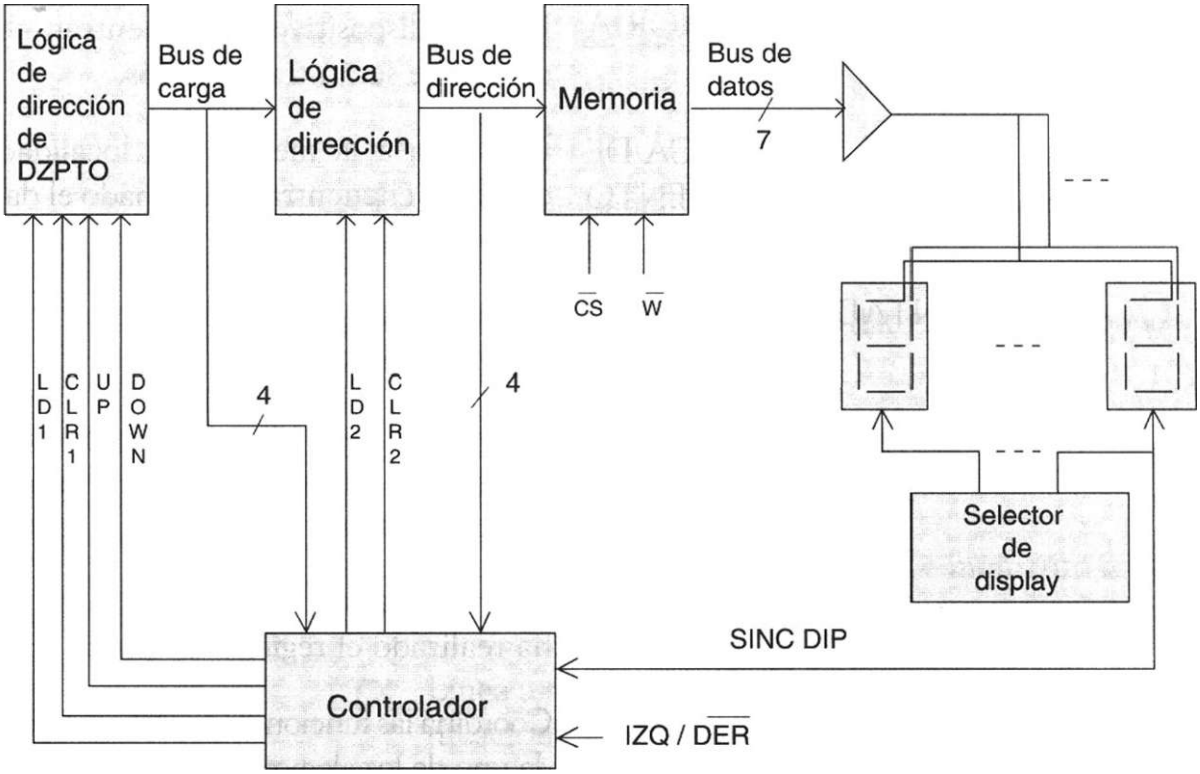
SINC DZP Señal de reloj para el controlador

BUS DE DIRECCIONES A través de este bus viaja la dirección de la localidad de memoria a leer.

BUS DE DATOS A través de este bus viaja el caracter leído de memoria y a visualizar en el display correspondiente.

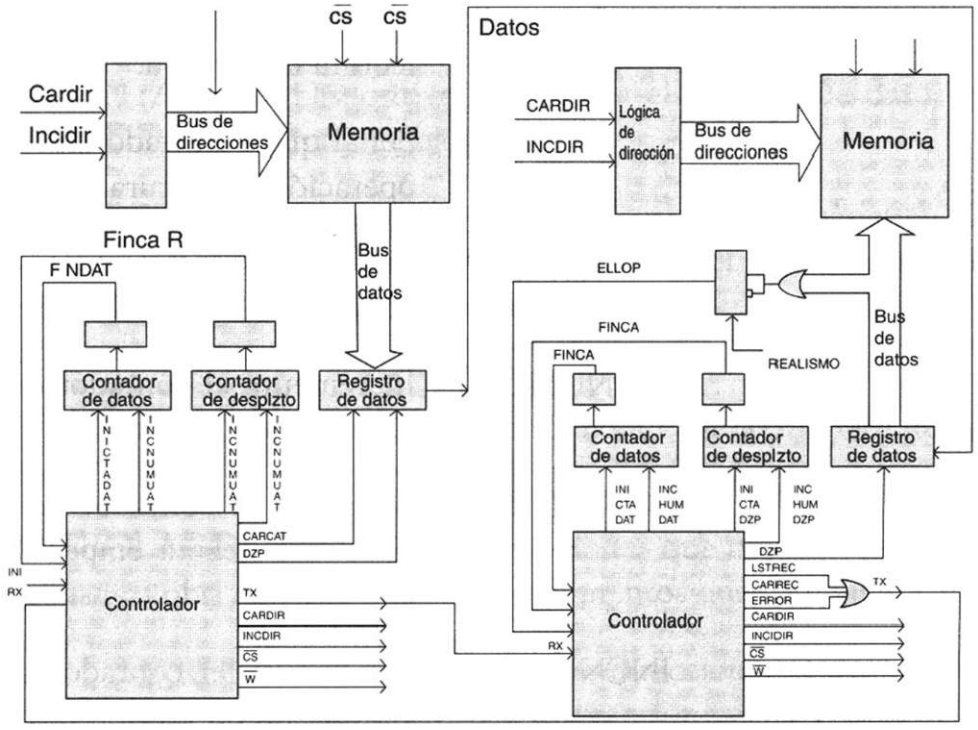
BUS DE CARGA A través de este bus viaja la dirección del primer caracter a leer cuando se realiza un desplazamiento de mensaje a la izquierda o derecha.

INTERMITENTE/FIJO Señal que controla la visualización intermitente o fija del mensaje mostrado.



Diseño de cartas ASM *mediante VHDL.*

6.19 Se desea diseñar un sistema transmisor-receptor de datos serie como el mostrado en la figura siguiente. La descripción de los elementos y señales de control correspondientes al trasmisor se muestran a continuación:



MEMORIA	En este dispositivo se encuentran almacenados los datos que serán transmitidos.
LOGICA DE DIRECCIONAMIENTO	Le indica a la memoria la localidad específica en dónde se encuentra almacenado el dato que se ha de transmitir.
CONTADOR DE DATOS	Dispositivo que indica el número de datos que se han transmitido.
REGISTRO DE DATOS	Dispositivo que almacena temporalmente el dato que se va a transmitir. El dato contenido en este registro se desplaza para realizar la transmisión en serie.
CONTADOR DE DESPLAZAMIENTO	Le indica al controlador cuántos desplazamientos ha realizado el registro de datos.
CONTROLADOR	Coordina las funciones que deberán de realizarse por cada uno de los elementos que forman el sistema.
CARDIR	Indica a la lógica de direccionamiento de memoria que debe cargar la dirección inicial a partir de la cual se encuentran almacenados los datos que van a ser transmitidos.
INCDIR	Señal de incremento de la dirección de memoria para leer el siguiente dato a transmitir.
CS	Habilitación de la memoria para las operaciones de lectura o escritura.
W	Señal que en estado alto indica a la memoria una operación de lectura.
INICTADA.	Indica al contador de datos en qué momento deberá comenzar a contar el número de datos transmitidos.
INCNUMDAT	Incrementa el contador de datos cada vez que un dato ha sido transmitido.
INICTADZP	Le indica al contador de desplazamientos en qué momento deberá empezar a contar el número de bits que han sido transmitidos.
INCNUMDZP	Incrementa el contador de desplazamientos cada vez que un bit ha sido transmitido.

CARDAT Le indica al registro de datos en que momento cargar el dato que se encuentra en el bus de datos de la memoria.

DZP Señal que habilita al registro de datos para desplazar (trasmitir) el dato contenido en él.

FINCAR Señal que indica cuando un dato ha sido trasmitido completamente.

FINDAT Señal que indica cuando un bloque de datos ha sido trasmitido.

INI Señal que indica al sistema en que momento se inicia la transmisión de datos.

TX Línea a través de la cual viajan las señales de comunicación del transmisor hacia el receptor.

RX Línea a través de la cual viajan las señales de comunicación del receptor hacia el transmisor.

La parte correspondiente al receptor se explica a continuación:

MEMORIA En este dispositivo se almacenarán los datos que se van a recibir.

LOGICA DE DIRECCIONAMIENTO Le indica a la memoria la localidad específica en donde se almacenará el dato a recibir.

CONTADOR DE DATOS Dispositivo que indica el número de datos que se han recibido.

REGISTRO DE DATOS Dispositivo que almacena temporalmente el dato que se va a recibir. Este registro desplaza el bit que se encuentra en su entrada serie, para la realización de la recepción serie.

CONTADOR DE DESPLAZAMIENTO Le indica al controlador cuando un dato ha sido recibido por completo.

CONTROLADOR Coordina las funciones que deberán realizarse por cada uno de los elementos que forman al sistema.

CARDIR Indica a la lógica de direccionamiento de memoria que debe cargar la dirección inicial a partir de la cual se almacenarán los datos que van a ser recibidos.

INCDIR Señal de incremento de la dirección de memoria, para almacenar el nuevo dato recibido.

CS Habilitación de la memoria para las operaciones de lectura o escritura.

W Señal que en estado bajo indica a la memoria una operación de escritura.

INICTADAT Indica al contador de datos en que momento deberá comenzar a contar el número de datos recibidos.

INCNUMDAT Incrementa el contador de datos cada vez que un dato ha sido recibido.

INICTADZP Le indica al contador de desplazamientos en que momento deberá empezar contar el número de bits recibidos.

INCNUMDZP Incrementa el contador de desplazamiento cada vez que un bit ha sido recibido.

DZP Señal que habilita al registro de datos para desplazar (recibir) el bit que se está transmitiendo.

FINCAR Señal que indica cuando un dato ha sido recibido.

FINDAT Señal que indica cuando un bloque de datos ha sido recibido.

TX Línea a través de la cual viajan las señales de comunicación del receptor al transmisor. Estas señales son:

LSTREC Listo a la recepción , indica que el sistema receptor está listo para recibir un dato.

CARREC Dato recibido, el sistema receptor indica que el dato que ha sido trasmitido fue recibido sin error y ha sido almacenado.

ERROR Esta señal se activa cada vez que ocurre un error de paridad cuando un dato ha sido recibido.

RX Línea a través de la cual viajan las señales de comunicación del trasmisor hacia el receptor.

ERROR Señal que en estado alto indica al controlador que hay un error de paridad (par o impar).

PARIDAD Señal que indica el tipo de paridad que deberán tener los datos recibidos.

Bibliografia

Floyd T.L.: *Fundamentos de Sistemas Digitales,* Prentice-Hall, 1998.

Stephen Brown, Zvonko Vranesic: *Fundamentals of Digital Logic Witch VHDL Design,* McGraw-Hill, 2000.

F.J. Hill and Peterson, *Computer Aid Logical Design Witch Emphasis on VLSI,* 4Th ed, Wiley, 1993.

C.H.Roth Jr., *Fundamentals of Logic Design,* West, 1993.

V.R Nelson, H.T. Nagle, *Digital Logic Circuit Analysis and Design,* Prentice-Hall,1995.

Charles H. Roth, Jr., *Digital Systems Design Using VHDL,* PWS Publishing Company, 1998.

Perry Douglas,VHDL, 2ed, New York, McGraw-Hill, 1994.

Herbert Taub, *Circuitos Digitales y Microprocesadores,* McGraw-Hill, 1983.

Capítulo 7

Diseño jerárquico en VHDL

Introducción

El diseño jerárquico es una herramienta de apoyo que permite la programación de extensos diseños mediante la unión de pequeños bloques; es decir, un diseño jerárquico agrupa varias entidades electrónicas, las cuales se pueden analizar y simular de manera individual con facilidad, para luego relacionarlas a través de un algoritmo de integración llamado Top Level (Fig. 7.1a).

La conceptualización del diseño Top Level difiere de la programación de los sistemas digitales, integrados en una entidad (Cap. 5). En este caso no se trata de integrar dos o más módulos electrónicos individuales (Fig. 7.1b), si no diseñar cada módulo por separado y luego coordinar su funcionamiento a través del algoritmo de programación Top Level.

Una ventaja importante del diseño jerárquico en la programación de grandes diseños es la facilidad para trabajar al mismo tiempo con otros diseñadores (paralelismo), ya que mientras uno puede diseñar una parte del sistema, otro puede desarrollar un bloque distinto para unirlos en un solo proyecto más tarde.

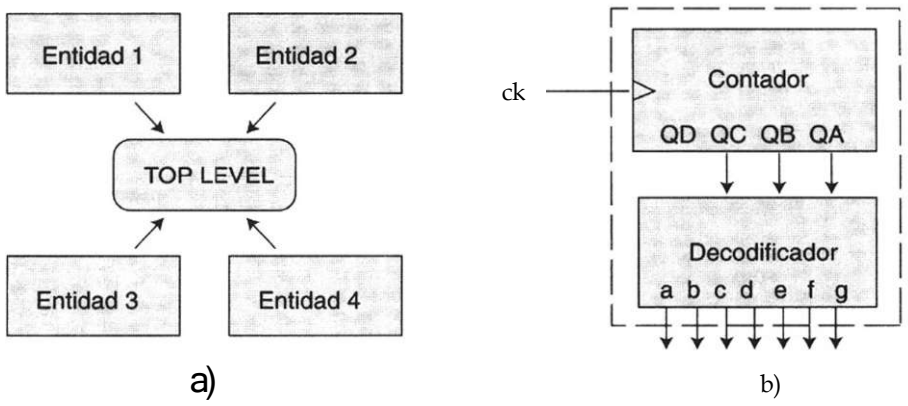


Figura 7.1 a) Estructuras jerárquicas, b) Integración de entidades.

7.1 Metodología de diseño de estructuras jerárquicas

Una metodología que se recomienda al programar extensos diseños es la siguiente:

1) Analizar con detalle el problema y descomponer en bloques individuales la estructura global.
2) Diseñar y programar módulos individuales (componentes).
3) Crear un paquete de componentes.
4) Diseñar el programa de alto nivel (Top Level).

Con el fin de describir y detallar la metodología empleada en el diseño de estructuras jerárquicas, consideremos el circuito AMD2909. Este dispositivo es un secuenciador de 4 bits desarrollado por la compañía Advanced Micro Devices, cuya función es transferir a su bus de salida (Y) una de entre cuatro fuentes internas y externas de datos. En la figura 7.2 se muestra la estructura externa del circuito y en la tabla 7.1 se indica la función de cada terminal.

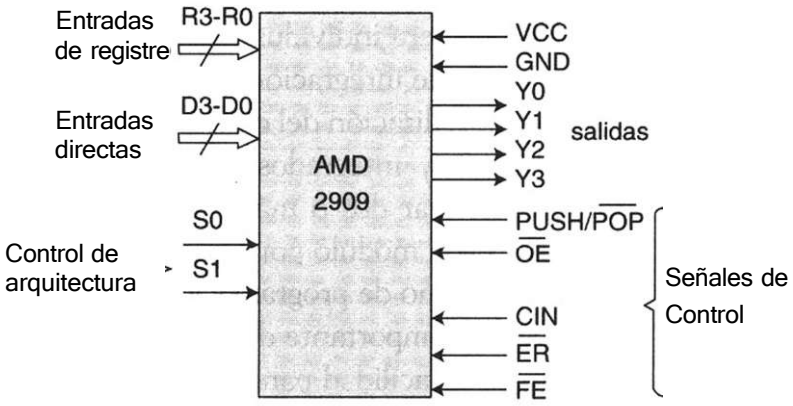


Figura 7.2 Secuenciador de cuatro bits AMD2909.

Terminal	Función
Entradas de registro R3-R0	Conservan el estado presente en una función de *hold o* retén.
Entradas directas D3-D0	Se usan como entradas del secuenciador para indicar un cambio de dirección en la lógica del programa.
Entrada /ER	Habilitación del registro R.
Entrada /FE	Habilitación del puntero de pila (stack pointer: ST).
Entrada CIN	Acarreo de entrada.
Entrada /OE	Habilitación de salidas.
Entrada PUSH/POP	Señales para brincos y retornos de subrutina.
Entradas SO y SI	Líneas de selección que determinan una de cuatro fuentes diferentes de entrada.
Salidas Y3-Y0	Salidas del secuenciador.
Salida COUT	Acarreo de salida.
Vcc y Gnd	Alimentación del circuito.

Tabla 7.1 Descripción de las señales de entrada y salida del secuenciador.

Descripción del circuito AMD2909

El secuenciador de 4 bits tiene entrada para un bus de datos externo D (D3 - *DO),* un registro *R (R3 - RO),* un apuntador de pila de una palabra ST y un sumador de 4 bits, que funciona como contador de programa (Fig. 7.3).

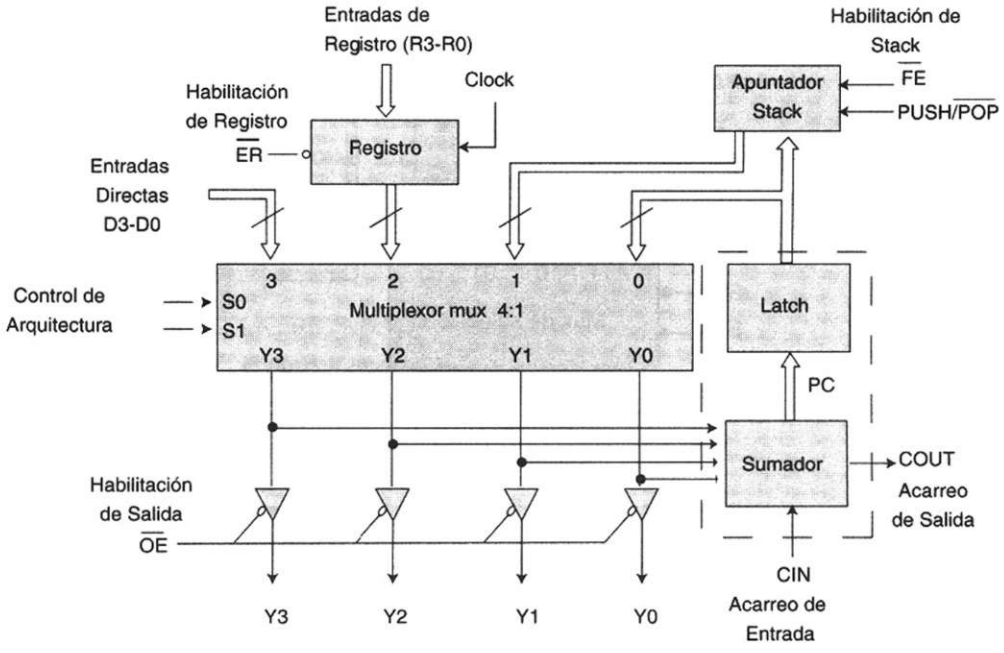


Figura 7.3 Descripción interna del circuito secuenciador AMD2909 [1].

Las entradas directas D se utilizan para canalizar las direcciones de carga desde una memoria de programa y/o control hacia el secuenciador de microprograma. En alguna de las arquitecturas, las entradas R sirven para almacenar el estado presente en una función de *hold* o retén. El contador de microprograma, *PC,* incrementa la salida en PC+ 1, siempre y cuando el acarreo de entrada *CIN* sea igual a 1. Esto permite realizar una instrucción de cuenta o almacenar en la pila la siguiente dirección cuando se hace un llamado a subrutina, por lo que al salir de ésta la dirección se obtiene de la pila y se canaliza hacia el bus de salida Y. Por último, el circuito contiene cuatro multiplexores de 4:1 que seleccionan una de sus cuatro entradas *PC, ST, R* y D, a través de sus líneas de selección *SO y SI.*

7.2 Análisis del problema y descomposición en bloques individuales de la estructura global

Como se mencionó, el diseño jerárquico basa su fortaleza en la descomposición o división de un diseño. Lo anterior permite analizar los diferentes subsistemas que lo forman y unirlos más tarde a través de un programa denominado Top Level.

En la figura 7.4 se muestra la interconexión interna de cada uno de los bloques y las señales de control que interactúan en cada uno de los subsistemas (multiplexor de selección, contador de microprograma, registro y apuntador de pila) del circuito AMD2909.

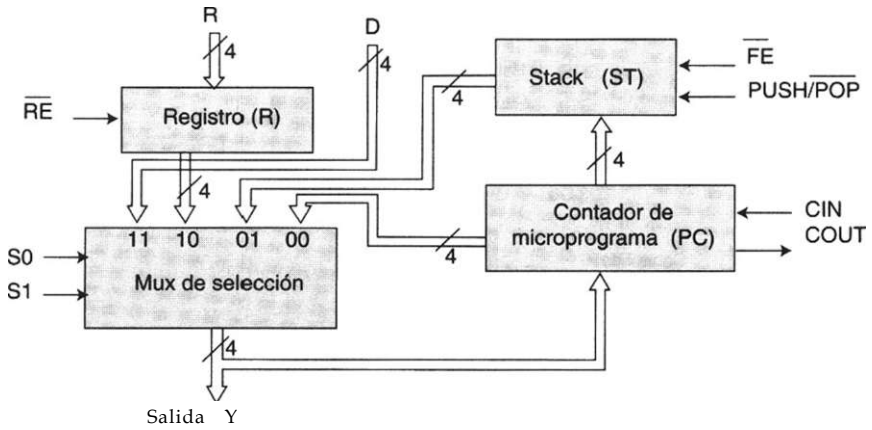


Figura 7.4 Diagrama general de diseño.

7.3. Diseño y programación de componentes o unidades del circuito

El primer paso de diseño consiste en programar de manera individual cada uno de los componentes y/o unidades del circuito. Un componente es la parte de un programa que define un elemento físico, el cual se puede usar en otros diseños o entidades. Con base en la figura anterior, iniciaremos la programación de cada uno de los bloques que forman el circuito AMD2909.

Diseño del registro (R)

En la figura 7.5 se muestra el registro R y las señales de control asociadas a él, mientras que en el listado 7.1 se puede observar el código VHDL correspondiente a esta entidad.

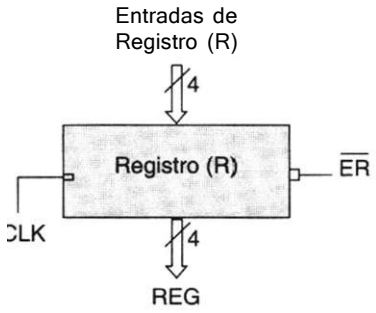


Figura 7.5 Registro de 4 bits.

```
-- Diseño del registro R
library ieee;
use ieee.std_logic_1164.all;
entity registro is port(
        R:    in std_logic_vector(3 downto 0);
 ER, CLK:     in std_logic;
     REG:     inout std_logic_vector(3 downto 0));
end registro;

architecture arq_reg of registro is
begin
 process (CLK,RE,REG,R) begin
   if  (CLK'event and CLK ='1')then
        if ER = '0' then
           REG <= R;
   else
           REG <= REG;
        end if;
   end if;
 end process;
end arq__reg ;
```

Listado 7.1 Programación de un registro de 4 bits.

Diseño del multiplexor

Este componente es simplemente un multiplexor cuádruple de 4:1, con dos líneas de selección (SO y *SI),* cuatro entradas *(R, ST, D, PC)* y una salida (Y) de 4 bits (Fig. 7.6).

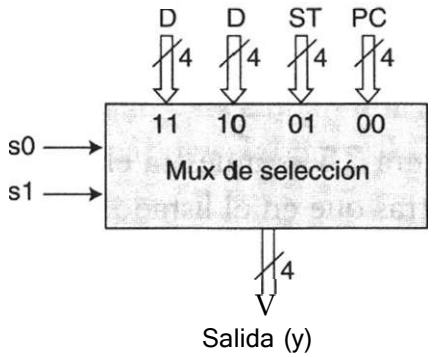


Figura 7.6 Multiplexor de selección.

El código de este componente se muestra en el listado 7.2.

```
-- Diseño del mux que selecciona una operación
library ieee;
use ieee.std_logic_1164.all;
   entity mux_4 is port(
      D,R,ST,PC: in std_logic_vector(3 downto 0);
              S: in std_logic_vector(1 downto 0);
              Y: out std_logic_vector(3 downto 0));
   end mux_4;

architecture arq_mux of mux_4 is
begin
    with S select
         Y <=   R when "00",
               ST when "01",
               PC when "10",
                D when others;
end arq_mux;
```

Listado 7.2 Código del multiplexor de selección.

Contador de microprograma (PC)

En esencia, la función de este bloque es incrementar la dirección de entrada cuando *CIN* es igual a uno. El diagrama correspondiente se encuentra en la figura 7.7.

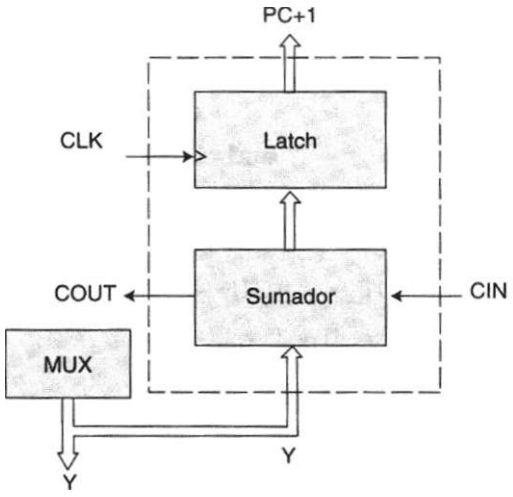


Figura 7.7 Contador de microprograma.

En el listado 7.3 se aprecia el código que describe el funcionamiento del contador de microprograma.

Como puede observarse, la manera más sencilla de programarlo es a través de la arquitectura funcional, considerando el circuito sumador y el contador como una entidad de diseño, con entradas y salidas generales. El funcionamiento es muy simple: cuando el acarreo de entrada *(CIN)* tiene el valor de '1' y el reloj del sistema está en la transición de '0' a '1', la dirección Y se incrementa en uno y su valor pasa al bus de salida PC. En caso contrario, cuando *CIN* = '0', se asigna a PC el valor de Y sin cambios.

```
-- Diseño del bloque de cuenta (contador de microprograma)
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
     entity mpc is port (
    CIN, CLK:    in std_logic;
            Y:   in std_logic_vector(3 downto 0);
         COUT:   inout std_logic;
           PC:   inout std_logic_vector(3 downto 0));
     end mpc;
```

Continúa

```
architecture arq_mpc of mpc is
begin
  process (CLK, Y, CIN) begin
            if (CLK'event and CLK = '1') then
                 if (CIN = '1') then
                     PC <= Y + 1;
                 else
                     PC<= Y;
                 end if;
            end if;
  end process;

   COUT <= (CIN and Y(0)and Y(1) and Y(2) and Y(3));

end arq_mpc;
```

Listado 7.3 Diseño del contador de microprograma.

Apuntador de pila (Stack Pointer)

La pila (stack) de la figura 7.8 está diseñada para almacenar un dato de 4 bits, de modo que cuando en un programa se hace un llamado a subrutina, la siguiente dirección (PC + 1) se almacena en la pila, por lo que al salir de la subrutina la dirección se toma de la pila y se envía al multiplexor de selección, el cual la canaliza al bus de salida (Y).

Para poder introducir y almacenar un dato en la pila, se debe habilitar primero la señal *FE* (habilitación de pila) y la señal PUSH (FE = '0' y PUSH=T). De esta forma, cuando la transición del reloj *(CLK)* sea de '0' a '1', el dato que se encuentra en PC se introduce y almacena en la pila hasta que la señal PUSH se deshabilite *(PUSH = '0').* Para sacar el dato se habilita la señal POP (POP = '0'), con lo que el dato se canaliza al bus de salida *ST.*

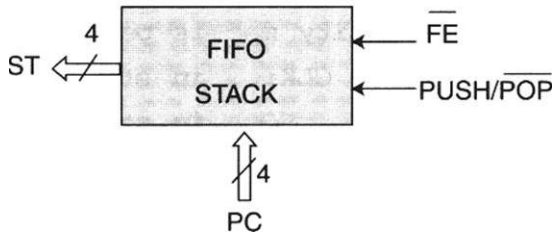


Figura 7.8 Pila de una palabra de 4 bits.

En el listado 7.4 se muestra el programa que indica el funcionamiento de la entidad correspondiente a la pila (stack).

```
— Diseño de una pila de una palabra de 4 bits
library ieee;
use  ieee.std_logic_1164.all;
use work.std_arith. all;

  entity stack is port(
   CLK,FE,PUSH,  POP:  in stdJLogic;
                PC:  in std_logic_vector (3 downto 0) ;
                ST:  inout std_logic_vector (3 downto 0));
    end stack;

architecture arq_stack of stack is
  signal var: std_logic_vector  (3 downto 0);
 begin
   process (FE, CLK, PUSH, POP, PC)
     variable x: std_logic_vector (3 downto 0);
       begin
           if (CLK'event and CLK = '1') then
                if (FE = '0') then
           if (PUSH = '1') then
                        x := PC; -- almacena dato
                        var <= x;
                   elsif (POP = '0') then
                        ST <= VAR; — saca dato
                    else
                        ST <= ST;
                   end if;
                end if;
            end if;
   end process;
end arq_stack;
```

Listado 7.4 Diseño de una pila de una palabra de 4 bits.

7.4 Creación de un paquete de componentes

Una vez que se ha diseñado cada módulo que forma la arquitectura general, se crea un programa que contenga los componentes de cada una de las entidades de diseño descritas con anterioridad. Para esto es necesario identificar primero el paquete en que se almacenarán los diseños (línea 4). En nuestro ejemplo el paquete recibe el nombre de *comps_sec* (el usuario escoge dicho nombre).

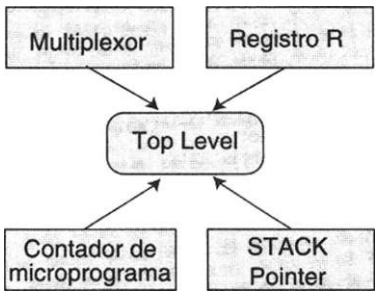
Como puede observar, en el listado 7.5 se ilustra la manera de declarar cada componente de un paquete llamado *comps_sec.* Note que cada componente se declara como una entidad de diseño, con la omisión de la palabra reservada is y agregando la cláusula component (líneas 5,11,18 y 24).

```
1 --Creación del paquete de componentes del secuenciador 2909
2 library ieee;
3 use ieee.std_logic_1164.all ;
4 package comps_sec is — declaración del paquete
5 component registro port(
6         R:       in std_logic_vector(3 downto 0);
7         ER,CLK:  in std_logic;
8         REG:     inout std_logic_vector(3 downto 0));
9   end component;
10
11 component mpc port( — declaración del contador
12  CIN,CLK:        in std_logic;
13        Y:        in std_logic_vector (3 downto 0) ;
14     COUT:        inout std_logic;
15       PC:        inout std_logic_vector(3 downto 0));
16   end component;

18   component stack port( -- declaración del stack
19      CLK,  FE,PUSH,POP:  in std_logic;
20       PC:    inout std_logic_vector(3 downto 0);
21       ST:    inout std_logic_vector(3 downto 0));
22   end component;

24    component mux_4 port( — declaración del multiplexor
25      D,R,ST,PC:  in std_logic_vector(3 downto 0);
26              S:  in std_logic_vector(1 downto 0);
27              Y:  buffer std_logic_vector (3 downto 0));
28   end component;
29 end comps_sec;
```

Listado 7.5 Creación del paquete que contiene los componentes del secuenciador.

7.5 Diseño del programa de alto nivel (Top Level)

Por último y como señalamos desde el principio, el algoritmo que une cada bloque o componente de diseño interconectado a través de señales o buses internos se denomina Top Level (Fig. 7.9).



a)

Figura 7.9 Programa de alto nivel (Top Level).

El programa de alto nivel que realiza esta función se muestra en el listado 7.6.

En la parte inicial del programa se llama a la librería *amd* (línea 1). Esta contiene el paquete *comps_sec,* que cuenta con los componentes que se utilizarán en el diseño (creación de librerías).

```
-- Diseño de los componentes del secuenciador 2909

1  library ieee, amd;
2  use ieee.std_logic_1164.all;
3  use work.std_arith.all;
4  use amd. comps_sec. all"; --paquete creado en la librería amd
5           entity amd2909 is port (
6          R:   in std_logic_vector (3 downto 0) ;
7          D:   in std_logic_vector (3 downto 0) ;
8         ER:   in std_logic;
9        CLK:   in std_logic;
10         S:   in std_logic_vector (1 downto 0) ;
11        FE:   in std_logic;
12      PUSH:   in std_logic;
13       POP:   in std_logic;
14       CIN:   in std_logic;
15      COUT:   inout std_logic;
16         Y:   buffer std_logic_vector (3 downto 0));
17     end amd2909;
```

Continúa

```
18
19 architecture arq_amd of amd2909 is
20  signal REG: std_logic_vector (3 downto 0);
21  signal ST:  std_logic_vector (3 downto 0);
22  signal PC: std_logic_vector (3 downto 0);

23  begin
24
25      -- inicia interconexión de los componentes
26
27  ul: registro port map (CLK => CLK, ER => ER, REG => REG, R => R);
28  u2: mpc    port map (CIN=>CIN, COUT=>COUT, CLK=>CLK, Y=>Y,
29                                PC=>PC);
30  u3: stack  port map (CLK =>CLK, SE => SE, PUSH=>PUSH, POP => 31
31                                POP, MPC => MPC, ST => ST) ;
32  u4: mux_4  port map (D=>D, R=>R, ST=>ST, PC=>PC, S=>S, Y=>Y);
33
34 end arq_sec;
```

Listado 7.6 Creación del programa principal.

La entidad se declara asignando las terminales de entrada y salida del secuenciador, las cuales se nombran tal como están referidas en su módulo (líneas 6 a 16). A continuación las señales que interconectan cada uno de los módulos se declaran dentro de la arquitectura (líneas 20 a 22). Note que estas señales no tienen asignada alguna terminal del dispositivo.

La segunda parte del código hace referencia a la conexión de los distintos componentes utilizando la cláusula port map (líneas 27 a 32). Además, u1, u2, u3 y u4 se denominan etiquetas de asignación inmediata. En cada asignación, el símbolo = > se usa para asociar (map) las señales actuales (es decir las que conforman la entidad amd2909) con las locales (los puertos que componen cada módulo de diseño). Una vez que se conecta cada módulo, el diseño se compila, generando un archivo .jed, que contiene todos los componentes creados.

Es importante resaltar en este programa la función de las terminales de entrada/salida (inout). Estas se utilizan para acoplar los diferentes bloques, debido a que son salidas de un bloque y entradas a un bloque diferente.

7.6 Creación de una librería en Warp

Como se puede apreciar, en el listado anterior (listado 7.6) se utilizó una librería llamada *amd,* la cual se creó para almacenar el paquete *comps_sec* que

contiene los componentes del diseño. Cabe mencionar que el uso de las librerías auxilia al programador en la reutilización de código; es decir, en la creación de un lugar para almacenar sus diseños, de modo que puedan servir para formar parte de otros diseños.

Ahora, con el fin de que pueda crear una librería de trabajo con facilidad, presentamos este apartado donde mostramos los pasos que se deben seguir con la herramienta Galaxy (Warp R4, Cypress Semiconductor).

1. Primero se genera un proyecto exclusivo para guardar los programas que formarán el diseño, o los que desean introducirse en la librería. Este proyecto (considerado como un archivo) debe llevar un nombre asignado por el usuario y la ruta (incluyendo la unidad de disco) en que se almacenará.
2. Para crear el proyecto, escoja del menú principal la opción project -> new y, de acuerdo con nuestro ejemplo, asigne los siguientes parámetros:

   c:\ejemplo\diseño

   donde *diseño* es el nombre del archivo en que se almacenarán todos los diseños.

3. Una vez creado, debe aparecer la ventana del proyecto en que se incluirán los diseños. Para esto elija la opción File -> Add del menú principal y los programas que formarán dicha aplicación (multiplexor, registro, pe y stack).

4. Si todos los diseños aparecen en la ventana de proyecto, tendrá que crear la librería en que se almacenarán. Para esto, seleccione la opción Libraries del menú File desde el menú principal. Esto abre la pantalla llamada M*anejador de Librerías en Galaxy.*

5. Con la ventana del manejador de librerías abierta, seleccione Create library desde la opción File (del menú principal). En la pantalla que se abre escriba el nombre con que identificará dicha librería —en nuestro caso tecleamos amd—. Presione Ok para aceptar los cambios.

6. Al llegar a este paso, el nombre de la librería creada debe aparecer en el cuadro libraries del manejador de librerías. Presione el botón Done para almacenar dicho nombre en el proyecto.

7. De nuevo en la pantalla principal, elija la opción Select all, del menú Files. Luego presione el botón File situado dentro de Synthesis options, con lo que aparecerá una ventana como la mostrada en la figura 7.10. El nombre en la parte superior de esta ventana debe ser el del archivo seleccionado de la Lista de diseños de la ventana de proyecto. Aquí elija la opción other y la librería de trabajo que ha creado (amd). A continuación presione Ok. Hay que repetir esta secuencia con cada uno de los diseños del proyecto.

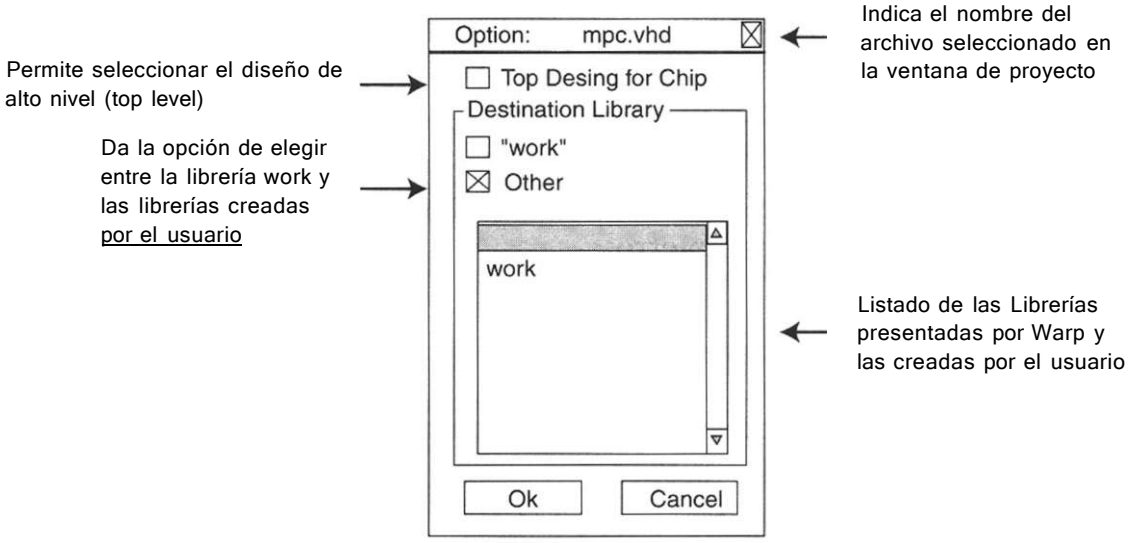

Option: mpc.vhd
Top Desing for Chip
Destination Library
"work"
Other
work
Ok
Cancel
Permite seleccionar el diseño de alto nivel (top level)
Da la opción de elegir entre la librería work y las librerías creadas por el usuario
Indica el nombre del archivo seleccionado en la ventana de proyecto
Listado de las Librerías presentadas por Warp y las creadas por el usuario

Figura 7.10 Archivos compilados dentro de la librería amd.

8. Una vez que se encuentren todos los diseños en la librería, escoja Save del menú File para asegurar que se guarden los cambios realizados.

9. El siguiente paso es compilar los diseños en la librería. Para esto se presiona el botón Smart de la ventaja de proyectos (project). Con esta opción todas las unidades de diseño y su paquete correspondiente se compilan en la librería.

10. Para comprobar la compilación de los diseños en la librería, se elige libraries del menú files. Esta opción abre el manejador de librerías de Galaxy. Aquí veremos que los diseños creados se encuentran dentro de la librería amd.

Para reafirmar la programación de las estructuras jerárquicas, consideremos el siguiente ejemplo.

Ejemplo 7.1 Use el diseño jerárquico para diseñar un microprocesador de 4 bits como el de la figura E7.1.

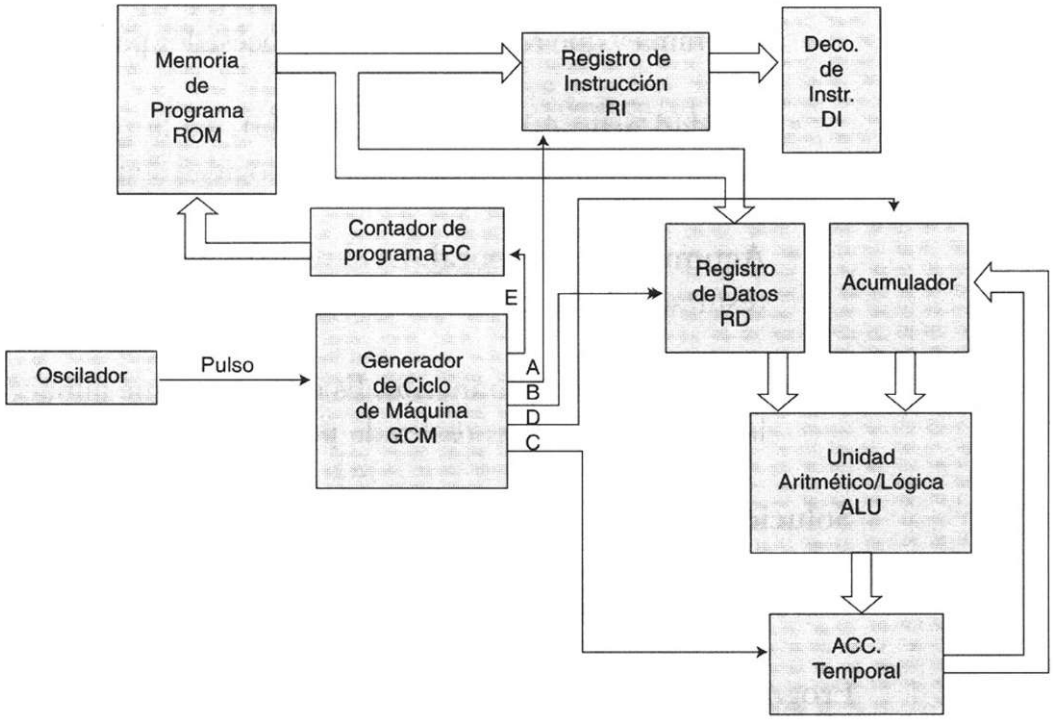


Figura E7.1 Microprocesador 4 bits.[1]

Descripción del microprocesador

En la figura E7.1 se aprecia cada uno de los bloques internos del microprocesador. Su función es la siguiente.

- Generador de ciclo de máquina. Este módulo es la unidad de control del microprocesador y su función es sincronizar y activar la participación de cada uno de los registros internos del microprocesador.
- Contador de programa. El contador es un registro interno del microprocesador que proporciona la siguiente dirección de memoria, sea para introducir un dato o una instrucción al microprocesador.

[1] Uruñuela José María. Microprocesadores, *Programación* e *Interfaces. McGraiv-Hill, pp 5-14*•

- Registro de instrucción. Almacena temporalmente la instrucción que se va a ejecutar en el microprocesador.

- Descodificador de instrucción. Es el elemento utilizado para interpretar y ejecutar la instrucción que se requiere realizar en la unidad aritmética y lógica.

- Registro de datos. Almacena los datos que provienen de la memoria de programa y que requiere el microprocesador para efectuar una operación.

- Unidad aritmética y lógica (ALU). Es la parte del microprocesador donde se realizan las operaciones lógicas o aritméticas.

- Acumulador temporal. Es el registro que almacena temporalmente el resultado de las operaciones realizadas dentro de la ALU.

- Acumulador permanente. Es el registro que almacena el resultado de la última operación realizada por el microprocesador.

Solución

I. Descomposición de la estructura global en bloques individuales

Programación del generador de ciclo de máquina. Este módulo coordina los procesos que realiza el microprocesador; utiliza cinco señales de control (A, B, C, D y E) que activan en orden secuencial los registros internos del microprocesador: *registro de instrucción (A), registro de datos (B), acumulador temporal (C), acumulador permanente (D) y contador de programa (E).* En la figura E7.2a) se muestra este módulo y el diagrama de tiempo de las señales de activación (E7.2b).

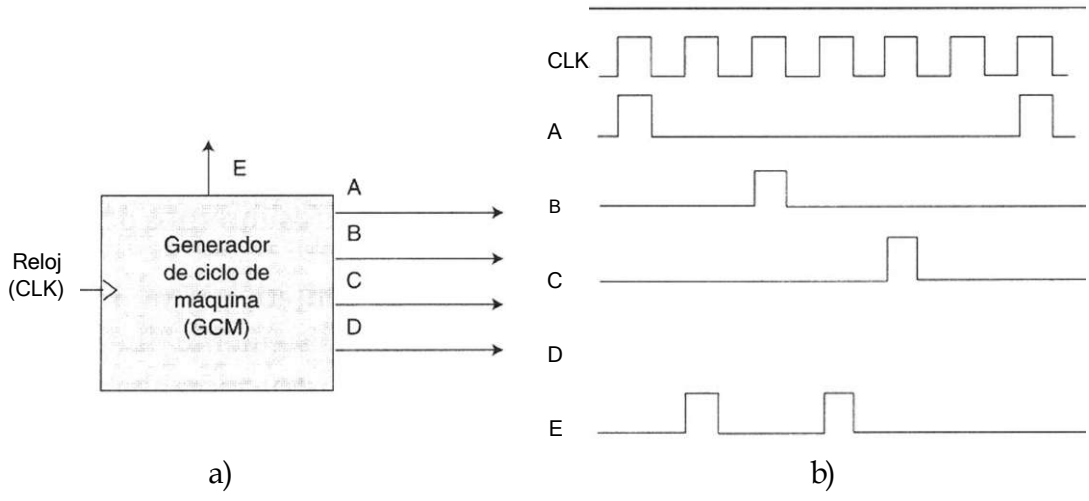


Figura E7.2 a) Generador de ciclo de máquina (GCM). b) Diagrama de tiempos.

La programación correspondiente a este bloque se muestra en el listado E7.1.

```
-- Módulo de generación ciclo de máquina
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity gem is port(
 CLK,RESET: in std_logic;
 A,B,C,D,E: inout std_logic;
 Q: inout std_logic_vector (3 downto 0));
end gem;
architecture a_gcm of gem is
begin
 process (elk,reset) begin
    if (elk'event and elk = '1') then
        Q <= Q + 1;
           if (RESET = '1" or Q = "0110") then
              Q <= "0000";
           end if;
     end if;
  end process;

 process (Q) begin
  case Q is
   when "0000" => A <= 'l';E <= '0'; B <= '0'; C <= '0'; D <= '0' ;
   when "0001" => A <= '0';E <= '1'; B <= '0'; C <= '0'; D <= '0';
   when "0010" => A <= '0';E <= '0'; B <= '1'; C <= '0'; D <= '0';
   when "0011" => A <= '0';E <= '1'; B <= '0'; C <= '0'; D <= '0';
   when "0100" => A <= '0';E <= '0'; B <= '0'; C <= '1'; D <= '0';
   when others => A <= '0';E <= '0'; B <= '0'; C <= '0'; D <= '1' ;
       end case;
  end process;
end a_gcm;
```

Listado E7.1 Programación del GCM.

En la programación de este bloque se integraron dos entidades: primero, la entidad de un contador del 0 al 5, el cual genera un número binario que determina la señal que se activará; en la segunda se define un circuito decodificador que al recibir el código binario procedente del contador, activa una y sólo una de las líneas de salida del generador ciclo de máquina.

Programación del registro de instrucción (RI). Este módulo almacena temporalmente las instrucciones provenientes de la memoria de programa. Su función es guardar el código binario de la instrucción mediante la señal de habilitación (A) que envía el generador de ciclo de máquina. En el listado E7.2 se muestra su programación.

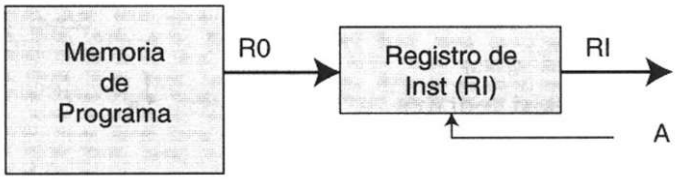


Figura E7.3 Registro de instrucción.

```
-- Módulo registro de instrucción
library ieee;
use ieee.std_logic_1164.all ;
entity reg_ins is port (
 A: inout std_logic;
 RO: inout std_logic_vector (3 downto 0);
 RI: inout std_logic_vector (3 downto 0));
end reg_ins;
architecture a_reg of reg_ins is
begin
 process (A,R0,RI) begin
   if (A'event and A = '1' ) then
        RI <= RO;
   end if;
 end process;
end a_reg;
```

Listado E7.2 Programación del registro de instrucción.

Programación del decodificador de instrucción (DI). La función de este bloque es convertir el código binario proveniente del registro de instrucción en una acción particular, la cual habilita una de varias operaciones lógicas o aritméticas dentro de la ALU. La programación de este bloque se muestra en el listado E7.3.

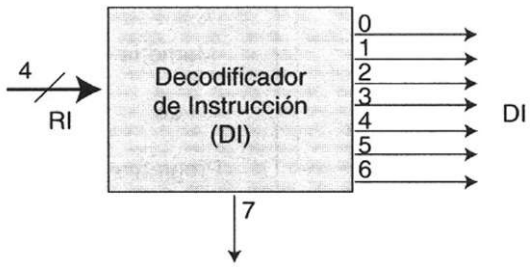


Figura E7.4 Decodificador de instrucción.

```
-- Módulo del decodificador de instrucción
library ieee;
use  ieee.std_logic_1164.all;
use  work.std_arith.all;
entity deco_ins is port (
 RI: inout std_logic_vector (3 downto 0);
 DI: inout std_logic_vector (0 to 7));
end deco_ins;
architecture a_deco of deco_ins is
begin
     process (RI) begin
      case RI is
        when "0000" => DI <= "10000000";-- función and
        when "0001" => DI <= "01000000";-- función or
        when "0010"=> DI <= "00100000";-- función xor
        when "0011" => DI <= "00010000";-- función suma aritmética
        when "0100" => DI <= "00001000";— función invertir el acum
        when "0101" => DI <= "00000100";-- función retén
        when "0110" => DI <= "00000010";-- función carga
        when "0111" => DI <= "00000001";-- función reset
        when others => DI <= "00000000";-- se inhabilita el DI
      end case;
  end process;
end a_deco;
```

Listado E7.3 Programación del decodificador de instrucción.

Programación del registro de datos (RD). Este módulo almacena temporalmente los datos provenientes de la memoria de programa. Su función es guardar el dato correspondiente mediante la señal de habilitación (B) que envía el generador de ciclo de máquina. En el listado E7.4 se muestra su programación.

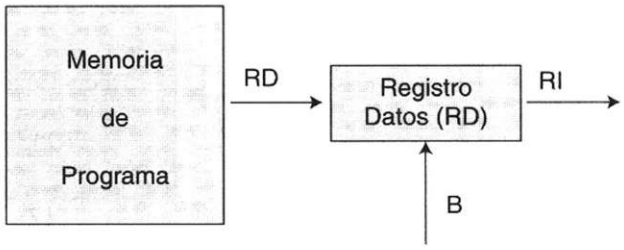


Figura E7.5 Registro de datos.

```
— Mòdulo del registro de datos
library ieee;
use ieee.std_logic_1164.ali;
entity reg_dat is port(
       B:  inout std_logic;
       RI: inout std_logic_vector (3 downto 0);
       RD: inout std_logic_vector (3 downto 0));
end reg_dat;
architecture a_dat of reg_dat is
begin
       process (B,RD,RI) begin
          if (B'event and B = '1') then
             RD <= RI;
          end if;
      end process;
end a_dat;
```

Listado E7.4 Programación del registro de datos.

Programación del contador de programa (PC). Sin duda un microprocesador requiere periféricos externos que le auxilien en su funcionamiento. En nuestro ejemplo, el contador de programa es un elemento que genera el bus de direcciones, ya sea para direccionar una memoria de programa (ROM), una memoria de datos (RAM) o ambas. En la figura E7.6 se muestra el diagrama correspondiente.

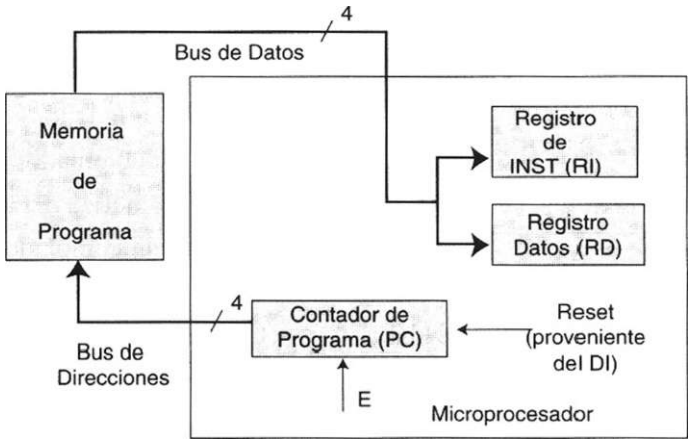


Figura E7.6 Conexión de una memoria externa.

Como puede observarse, el contador se incrementa cada que se genera la señal (E) proveniente del generador ciclo de máquina. La programación de esta unidad se muestra a continuación (listado E7.5).

```
— Módulo contador de programa
library ieee;
use  ieee.std_logic_1164.all;
use work.std_arith.all ;
entity pcount is port (
       E: inout s td_log i c ;
      DI:  inout std_logic_vector  (0  to  7);
      PC:  inout std_logic_vector  (3  downto  0));
end pcount;
architecture a_pc of pcount is
begin
 process  (E)  begin
    if  (E'1' event and E = '1') then
         PC <= PC + 1;
           if  (DI = "00000001")  then
             PC <=  "0000";
           end if ;
    end if ;
 end process ;
end a_pc ;
```

Listado E7.5 Programación del contador de programa.

Programación de la unidad aritmética y lógica (ALU). Tal como su nombre indica, la función de este bloque es realizar las operaciones aritméticas y lógicas del microprocesador. Según se aprecia en la figura E7.6, la ALU de nuestro ejemplo puede llevar a cabo ocho operaciones, las cuales se refieren por medio de un código de operación de cuatro bits (tabla E7.1).

Código de Operación	Instrucción
0000	AND, el acumulador con el dato inmediato
0001	OR, el acumulador con el dato inmediato
0010	XOR, el acumulador con el dato inmediato
0011	Suma aritmética del acumulador con el dato inmediato
0100	Invertir el acumulador
0101	Retener el dato (hold)
0110	Cargar un dato en el acumulador
0111	Brincar a cero

Tabla E7.1 Descripción de las operaciones del microprocesador

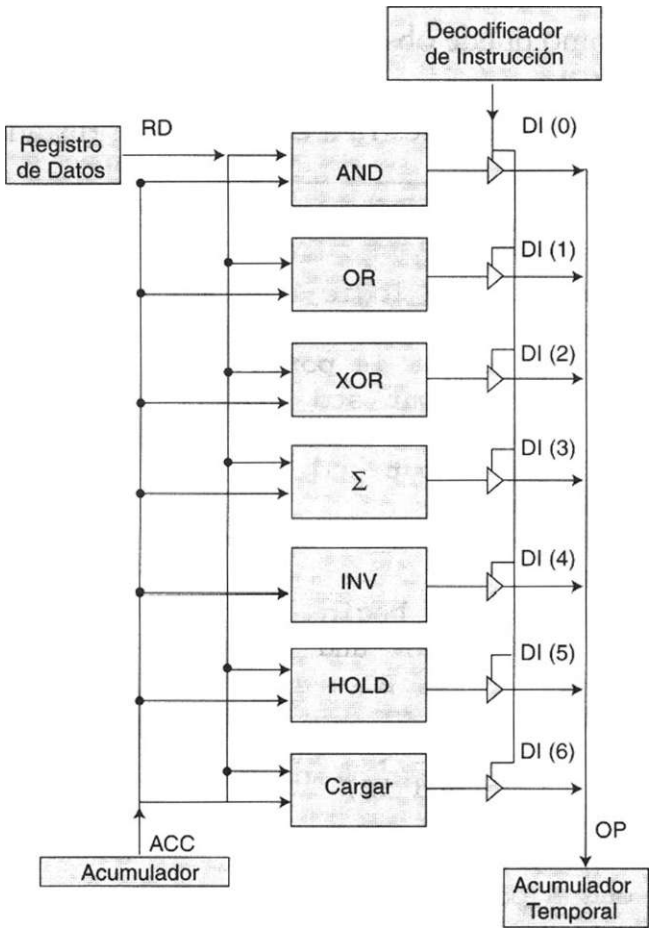


Figura E7.7 ALU: unidad aritmética lógica.

Cabe mencionar que seis de las ocho operaciones que realiza la ALU requieren dos datos para funcionar: uno se almacena con anterioridad en el acumulador y el otro proviene del registro de datos. Observe que la única operación que no requiere dos datos es la función de "invertir", ya que se realiza invirtiendo el contenido del acumulador.

Note también que a la salida de cada bloque de operación (and, or, xor, etc.) se encuentra un buffer triestado activo en alto que, con ayuda del decodificador de instrucción, habilita una de las siete salidas correspondientes a cada operación. Observe que el resultado de las operaciones se almacena de nuevo en el acumulador. En este caso utilizamos una señal auxiliar llamada OP (operación), la cual guardará temporalmente el resultado de dicha operación y luego lo canalizará al acumulador temporal. El listado correspondiente se muestra a continuación, listado E7.6.

```
-- Programación de la unidad aritmética y lógica
library ieee;
use  ieee.std_logic_1164.all;
use work.std_ari th.all;
entity alu is port (
      DI: inout std_logic_vector (0 to 7) ;
      RD: inout std_logic_vector (3 downto 0);
     ACC: inout std_logic_vector (3 downto 0);
      OP: inout std_logic_vector (3 downto 0));
end alu;
architecture a_alu of alu is
begin
 process (DI,ACC,RD) begin
    if (DI = "10000000") then
        OP <= ACC and RD;
   elsif (DI = "01000000") then
        OP <= ACC or RD;
   elsif (DI = "00100000") then
        OP <= ACC xor RD;
   elsif (DI = "00010000") then
        OP <= ACC + RD;
   elsif (DI = "00001000") then
        OP <= not ACC;
   elsif (DI = "00000100") then
        OP <= ACC; -- HOLD
   else
        OP <= RD;--CARGAR ACUMULADOR
   end if;
end process;
end a alu;
```

Listado E7.6 Programación de la unidad aritmética y lógica.

Programación del acumulador temporal (ACT). La función de este módulo es almacenar temporalmente el resultado proveniente de la ALU (OP) y después canalizarlo por medio de su salida (ACT) al acumulador permanente; este dato se almacena mediante su señal de habilitación (C) proveniente del generador ciclo de máquina. En el listado E7.7 se muestra la programación de este módulo.

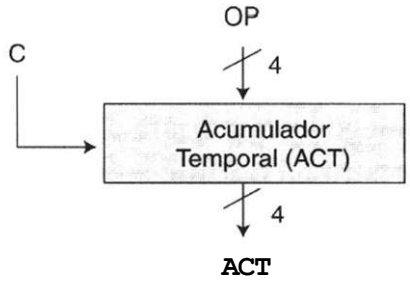


Figura E7.8 Decodificador de instrucción.

```
—Programación del acumulador temporal
library ieee;
use ieee.std_logic_1164.all ;
entity acct is port (
        C: inout s td_logic;
       OP: inout std_logic_vector (3 downto 0);
      ACT: inout std_logic_vector (3 downto 0));
end acct;
architecture a_acct of acct is
begin
      process (C) begin
        if (Cevent and C= '1') then
           ACT <= OP;
        end if;
       end process ;
end a_acct;
```

Listado E7.7 Acumulador temporal.

Programación del acumulador permanente (ACC). La función de este módulo es almacenar el resultado final de la última operación realizada por la ALU, ya sea para enviarlo como aplicación externa o retroalimentar al microprocesador. Este dato se almacena mediante su señal de habilitación (D) proveniente del generador de ciclo de máquina. En el listado E7.8 se muestra la programación de este bloque.

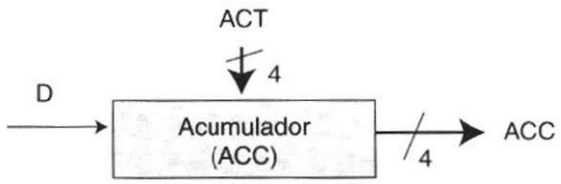


Figura E7.9 Acumulador

```
-- Módulo del acumulador permanente
library ieee;
use ieee.std_logic_1164.all ;
entity acum is port !
      D: inout std_logic;
   ACT: inout std_logic_vector (3 downto 0);
   ACC: inout std_logic_vector (3 downto 0));
end acum;
architecture a_acc of acum is
begin
      process (D) begin
         if (D' event and D= '1') then
            ACC <= ACT;
        end if ;
     end process ;
end a_acc;
```

Listado E7.8 Programación del acumulador permanente.

II. Programación del paquete de componentes

A continuación (listado E7.9) se presenta el paquete que incluye los componentes del sistema: *generador de ciclo de máquina* (gcm), *registro de instrucción*

(reg_ins), decodificador de instrucción (deco_ins), registro de datos (regjdat), contador de programa (pcount), unidad aritmética lógica (alu), acumulador temporal (act) y acumulador (acum.)

```
library ieee;
use ieee.std_logic_1164.all;
package micro4 is

    component gem port (
          CLK,RESET: in std_logic;
          A,B,C,D,E: inout std_logic;
                  Q: inout std_logic__vector (3 downto 0));
    end component;

    component reg_ins port(
           A: inout std_logic;
          RO: inout std_logic_vector (3 downto 0) ;
          RI: inout std_logic_vector (3 downto 0));
    end component;

     component reg_dat port(
           B: inout std_logic;
          RI: inout std_logic_vector (3 downto 0);
          RD: inout std_logic_vector (3 downto 0));
     end component;

     component deco_ins port(
          RI: inout std_logic_vector (3 downto 0);
          DI: inout std_logic_vector (0 to 7));
      end component;

      component alu port(
          DI: inout std_logic_vector (0 to 7) ;
          RD: inout std_logic_vector (3 downto 0);
         ACC: inout std_logic_vector (3 downto 0) ;
          OP: inout std_logic_vector (3 downto 0));
      end component;

      component pcount port (
           E: inout s t d_log i c;
          DI: inout std_logic_vector (0 to 7);
          PC: inout std_logic_vector (3 downto 0));
      end component;
```

Continúa

```
    component acct port(
        C: inout std_logic;
       OP: inout std_logic_vector (3 downto 0);
      ACT: inout std_logic_vector (3 downto 0) ) ;
    end component;

    component acum port(
        D: inout std_logic;
      ACT: inout std_logic_vector (3 downto 0);
      ACC: inout std_logic_vector (3 downto 0));
    end component;
end micro4;
```

Listado E7.9 Paquete de componentes.

III. Programa de alto nivel

En el listado E7.10 se muestra el programa que conecta todos los bloques *(Top Level).* Observe que a diferencia del ejemplo del secuenciador, el paquete *micro4* no se creó en una librería nueva, sino en la librería work, lo cual también es una acción permitida en VHDL.

```
library ieee;
use  ieee.std_logic_1164.all;
use work.std_arith.all ;
use  work,micro4.all;
entity mp4 is port  (
   CLK,RESET: in std_logic; — reloj y reset para el gem
         R0: inout std_logic_vector (3 downto 0); --entrada de dato
         RI: inout std_logic_vector (3 downto 0); --entrada de inst
        ACC: inout  std_logic_vector (3 downto 0)); —salida del dato
end mp4;
architecture a_mp4 of mp4 is
signal A,B,C,D,E:  std_logic;
signal Q,ACT,0P,PC:  std_logic_vector (3 downto 0);
signal DI:  std_logic_vector (0 to 7);
signal RD:  std_logic_vector (3 downto 0);
 begin
```

Continúa

```
U1: GCM port map (CLK => CLK, RESET => RESET, A => A, B => B, C => C,
                  D => D, Q => Q);
U2: REG_INS port map (A => A, RO => RO, RI => RI);
U3: REG_DAT port map (B => B, RI => RI, RD => RD) ;
U4: DEC0_INS port map (RI => RI, DI => DI);
U5: ALU port map (DI => DI, RD => RD, ACC => ACC, OP => OP);
U6: PCOUNT port map (E => E, DI =>DI, PC => PC);
U7: ACCT port map (C => C, OP => OP, ACT => ACT);
U8: ACUM port map (D => D, ACT => ACT, ACC => ACC);

end a_mp4;
```

Listado E7.10 Programa de alto nivel.

Ejercicios

Diseño Top Level

7.1 A continuación se describe el funcionamiento de un sistema global de un contador de llenado de pastillas.[1]

Inicialización del sistema

Antes que nada el sistema debe inicializar para arrancar en los estados binarios correctos. En primer lugar, el contador y los registros se deben poner a cero, para que la cuenta empiece a partir de cero. A continuación, el número de pastillas (hasta 99) que se introducirá en cada bote, se introduce a través del teclado numérico, el cual se convierte a BCD mediante el codificador y se almacena en el registro A. En la a) el número decimal 75 se introduce a través del teclado numérico, lo que indica que en cada bote se introducirán 75 pastillas. Cada dígito tecleado se codifica en BCD y se envían, los 4 bits a la vez, al registro A, para ser almacenado. El código BCD de 8 bits se coloca en las líneas de salida del registro y se aplica al decodificador, el cual lo convierte de BCD a código de 7 segmentos para iluminar el display.

Simultáneamente, el mismo código BCD se aplica al convertidor de código, el cual lo convierte a binario. El número binario resultante, 75, se presenta a la entrada A del comparador. Ahora el sistema está inicializado y listo para empezar a contar pastillas.

El sistema en proceso

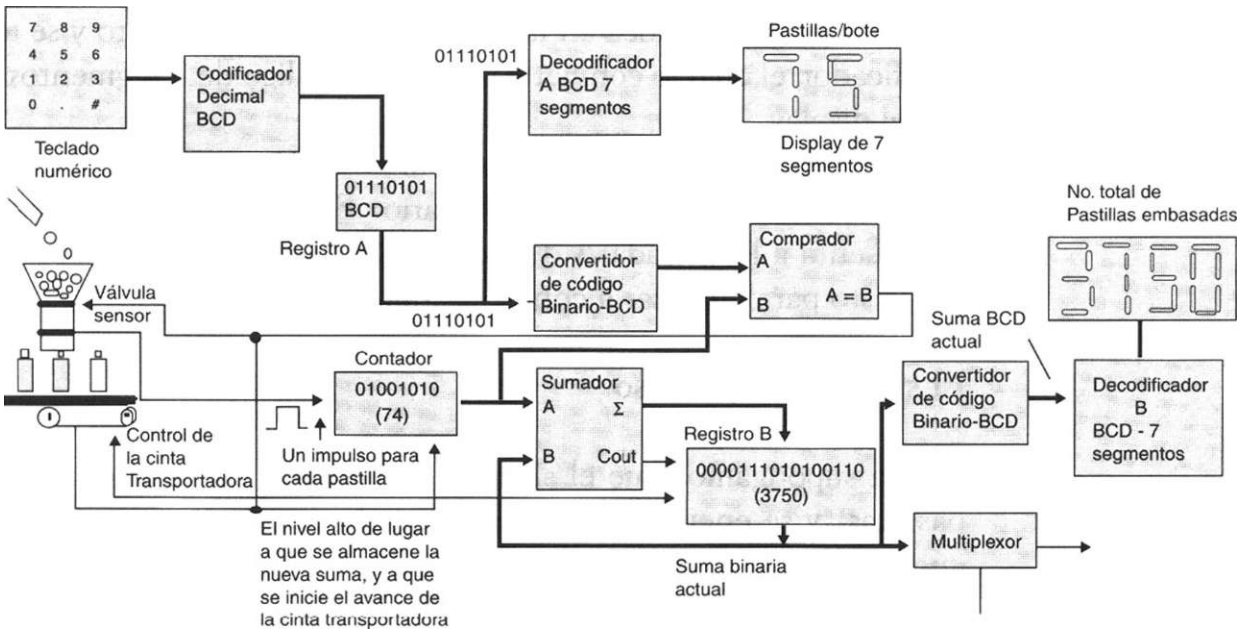
Ahora supongamos que el sistema ha contado un total de 50 botes (3750 pastillas) y se encuentra en el proceso de contar pastillas para el bote cincuenta y uno. En realidad, como indica el contenido del contador de la figura a), ha contado 74 pastillas. Observe que el registro B mantiene la suma binaria anterior (3750) que se ha pasado a BCD y luego a formato de 7 segmentos para el display. El registro B se actualiza cada vez que se llena un bote, pero no mientras se está llenando.

La salida del comparador está a nivel bajo (LOW, 0) , ya que las entradas A y B no son iguales en ese instante. En la entrada A hay un 75 binario, y en la entrada B un 74 binario. La salida del sumador siempre indica la suma más reciente, en este caso, 3824, pero este número no se almacena en el registro B hasta que el bote esté lleno.

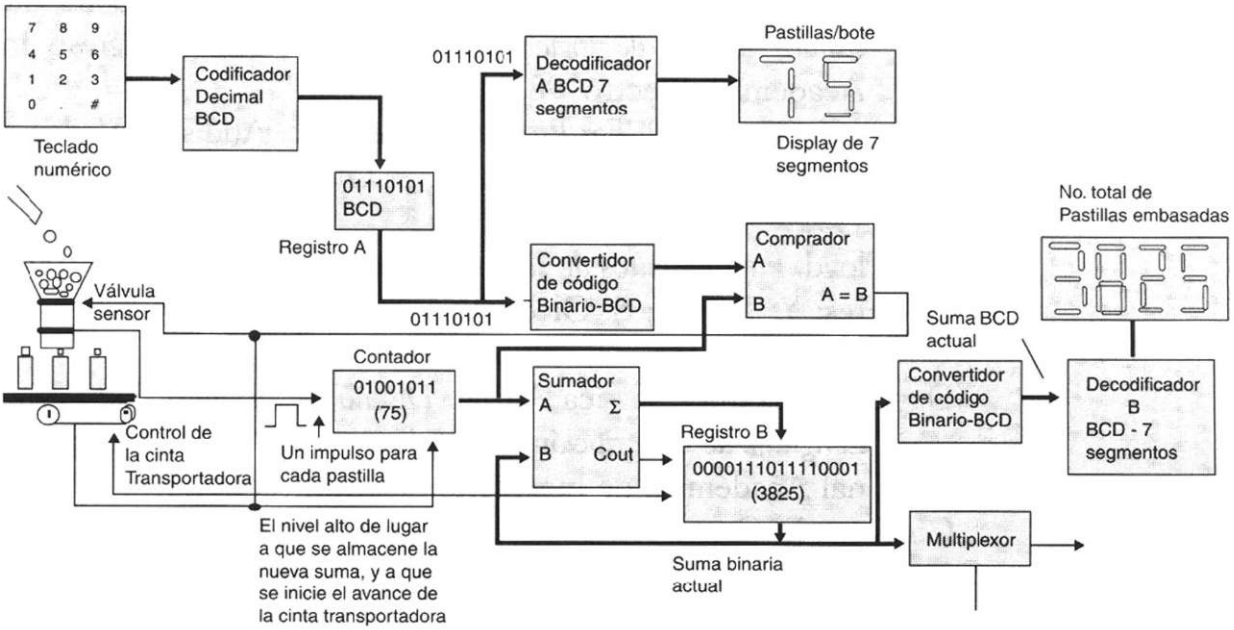
[1] Síntesis de un circuito secuencial síncrono, T.L. Floyd: *Fundamentos de Sistemas Digitales.* Prentice Hall, 1998, p. 519.

En la figura b) se muestra el estado siguiente donde el contador ha alcanzado el valor binario de 75, por lo que la salida A = B del comparador pasa a estado alto (HIGH, 1), lo que da lugar a que ocurran varias cosas simultáneamente: se detiene el flujo de pastillas, permitiendo que la nueva suma binaria 3825 -figura b)- a la salida del sumador se almacene en el registro B, y se inicia un avance de la cinta transportadora para colocar el siguiente bote en posición. El número binario almacenado en el registro B, que es ahora la suma actual, se aplica al convertidor de código BCD-7 segmentos, y en formato de 7 segmentos se actualiza el display con el valor 3825. Tan pronto como se coloca el siguiente bote, el contador se pone a cero y el nuevo ciclo comienza.

a) Jerarquice el problema e identifique entradas y salidas de cada bloque
b) Programe mediante top level el sistema general



a) Sistema de embasado de pastillas.



b) Sistema de embasado de pastillas (estado siguiente).

Bibliografía

G. Maxinez David, Alcalá J. Jessica. *Diseño de secuenciadores integrados utilizando campos de lógica programable FPGA.* Congreso Internacional Académico Electro'98. México, 1998.

Skahill Kevin. *VHDL For Programmable Logic.* Addison Wesley, 1996.

Mazor Stanley, Langstraat Patricia. A *Guide to VHDL.* Kluwer Academic, 1992.

T.L. Floyd. *Fundamentos de sistemas digitales.* Prentice Hall, 1998.

Ll. Teres; Y. Torroja; S. Olcoz; E. Villar. *VHDL, lenguaje estándar de diseño electrónico.* McGraw-Hill, 1998.

David G. Maxinez, Jessica Alcalá. *Diseño de Sistemas Embebidos a través del Lenguaje de Descripción en Hardware VHDL.* XIX Congreso Internacional Académico de Ingeniería Electrónica. México, 1997.

C. Kloos, E. Cerny. *Hardware Description Language and their aplications. Specification, Modelling, Verification and Synthesis of Microelectronic Systems.* Chapman & Hall, 1997.

IEEE: *The IEEE standard VHDL Language Reference Manual.* IEEE-Std-1076-1987, 1988.

Zainalabedin Navabi. *Analysis and Modeling of Digital Systems.* McGraw-Hill, 1988.

P J. Ashenden. *The Designer's guide to VHDL.* Morgan Kauffman Publishers, Inc., 1995.

R. Lipsett, C. Schaefer. *VHDL Hardware Description and Design.* Kluwer Academic Publishers, 1989.

J. R. Armstrong y F. Gail Gray. *Structured Design with VHDL.* Prentice Hall, 1997.

J. A. Bhasker. A *VHDL Primer.* Prentice Hall, 1992.

H. Randolph. *Applications of VHDL to Circuit Design.* Kluwer Academic Publisher.

Referencias

[1J P Hayes John. *Diseño de sistemas digitales y microprocesadores.* McGraw-Hill, 1988.

[2] José María Uruñuela. *Microprocesadores, programación e interfaces.* McGraw-Hill, 1990.

Capítulo 8

Sistemas embebidos en VHDL

Introducción

Los microprocesadores tienen un gran uso como componentes fundamentales en sistemas electrónicos modernos, ya que incrementan su capacidad y proporcionan una mejor ejecución en las tareas que se encomiendan reduciendo su costo. Cuando un sistema electrónico incorpora un microprocesador, ofrece herramientas de control adicionales con tan sólo un incremento relativo en el costo de software de desarrollo. Debido a esto cada semana millones de pequeños chips de computadora salen de fábricas como Motorola, National, Mitsubishi, etcétera y encuentran diversas y nuevas aplicaciones en productos de uso común como sistemas de transporte, producción de comida, defensa militar, sistemas de comunicación, etcétera.

8.1 Sistemas embebidos

Estos sistemas electrónicos basados en microprocesadores y diseñados para aplicaciones específicas se les conoce como sistemas embebidos, es decir, se trata de un sistema cuyo hardware y software se diseñan específicamente optimizando su funcionamiento para resolver uno o varios problemas con eficiencia. El término embebido se refiere al hecho de que la microcomputadora es encapsulada en un solo circuito. Estos sistemas interactúan con todo lo que los rodea y funcionan como el monitor o el control de algún proceso. Este hardware suele diseñarse a nivel de componente en un circuito integrado o de tarjeta; por otra parte, el software que controla al sistema se programa en una memoria de sólo lectura (ROM) y no es accesible para el usuario del dispositivo.

Los sistemas *embebidos* siempre están compuestos por microprocesador, memoria, entradas y salidas a periféricos y un programa de aplicación dedicado, guardado permanentemente en la memoria.

Un programa de aplicación para un sistema embebido forma parte del sistema y por lo general se ejecuta sin necesidad de un sistema operativo; por tanto, la aplicación del programa ha de incluir el software para controlar e interactuar con los dispositivos periféricos del sistema.

En la actualidad estos sistemas están presentes en la vida diaria en lavadoras, videos, televisores, calefacción, sistemas de alarma, procesadores de alimentos, computadoras, etc. Además, van de lo más simple a lo muy complejo, ya que controlan desde las luces en los zapatos tenis hasta los sistemas de control de los aviones militares; asimismo, cuentan con varias entradas y salidas a otros sistemas o periféricos que son dispositivos conectados al sistema principal y controlados por el microprocesador. Los microprocesadores que controlan este tipo de sistemas se pueden dividir en:

- *Microprocesadores individuales.* Se encuentran en dispositivos como sensores de temperatura, detectores de humo y gas, interruptores de circuitos [1].
- *Conjuntos de microprocesadores sin funciones de reloj*. Se hallan en controladores de flujo, amplificadores de señal, sensores de posición, servomecanismos de válvula [1].
- *Conjuntos de microprocesadores con funciones de reloj.* Se encuentran en equipos médicos de monitoreo, controladores, centrales telefónicas, sistemas de adquisición de datos (SCADA), sistemas de diagnóstico y tipo real.
- *Sistemas computarizados usados en control de procesos e industrias.* Son computadoras conectadas a equipos para controlarlos[l].

Campo	Aparatos	Función de ejecución del sistema
Hogar	Lavadora	Control del agua y ciclo de giros
	Control remoto	Acepta la señal de las teclas y manda pulsos infrarrojos al sistema base
	Despertadores y relojes	Controla el tiempo, la alarma y la pantalla
	Juegos y juguetes	Control del bastón de mando (*joystick*) y salida de video
	Audio/video	Interactúa con el operador
Comunicación	Contestadoras telefónicas	Reproductor de mensajes de salida; guarda y organiza mensajes
	Teléfonos celulares	Teclado de entrada, sonido de entrada y salida, comunicación con la estación central

Tabla 8.1 Aplicaciones de los sistemas embebidos[1]

[1] Tomado de Ganssle J. (1999).

Campo	Aparatos	Función de ejecución del sistema
Automóviles	Freno automático	Optimiza el frenado y la estación de parada
	Encendido eléctrico	Controla el encendido y la mezcla de los inyectores de combustible
Militar	Armas pequeñas	Reconoce el blanco
	Sistemas de misiles	Orden programada de forma directa al blanco deseado
	Sistema de posición global	Determina el punto exacto donde se encuentra el usuario
Industrial	Sistemas de control de tráfico	Detecta la posición del carro y controla la señalización (semáforos)
	Sistemas robot	Controla los motores para el posicionamiento de brazos, etcétera.
	Código de barras lectura y escritura	Controla tarcas a partir de valores detectados por sensores
		Realiza lectura de valores de entradas de escritura para control de inventarios
Médico	Monitor cardiaco	Monitorea funciones cardiacas
	Tratamiento de cáncer	Controla radiación, administración de sustancias químicas o calor

Tabla 8.1b Continuación

8.1.1 Diseño de sistemas embebidos

Para tener una estructura fundamental común, los sistemas embebidos cuentan con un ciclo de desarrollo representado por el proceso en que se generó el sistema. En este ciclo de desarrollo se muestran los avances en la ingeniería para maximizar la posibilidad de producir un sistema efectivo confiable y al que se puede dar mantenimiento.

Cada fase de ciclo de desarrollo se identifica mediante un bloque con una actividad dominante y el resultado primario de cada fase se presenta en la salida de bloque. Cada resultado está representado por documentación que describe al sistema en esa etapa. Como consecuencia, se tiene una documentación completa y exacta de cada fase de desarrollo y del sistema final. El mismo ciclo es aplicable a un sistema diseñado para una persona o un equipo. En la figura 8.1 se muestra el ciclo de desarrollo de un sistema embebido.

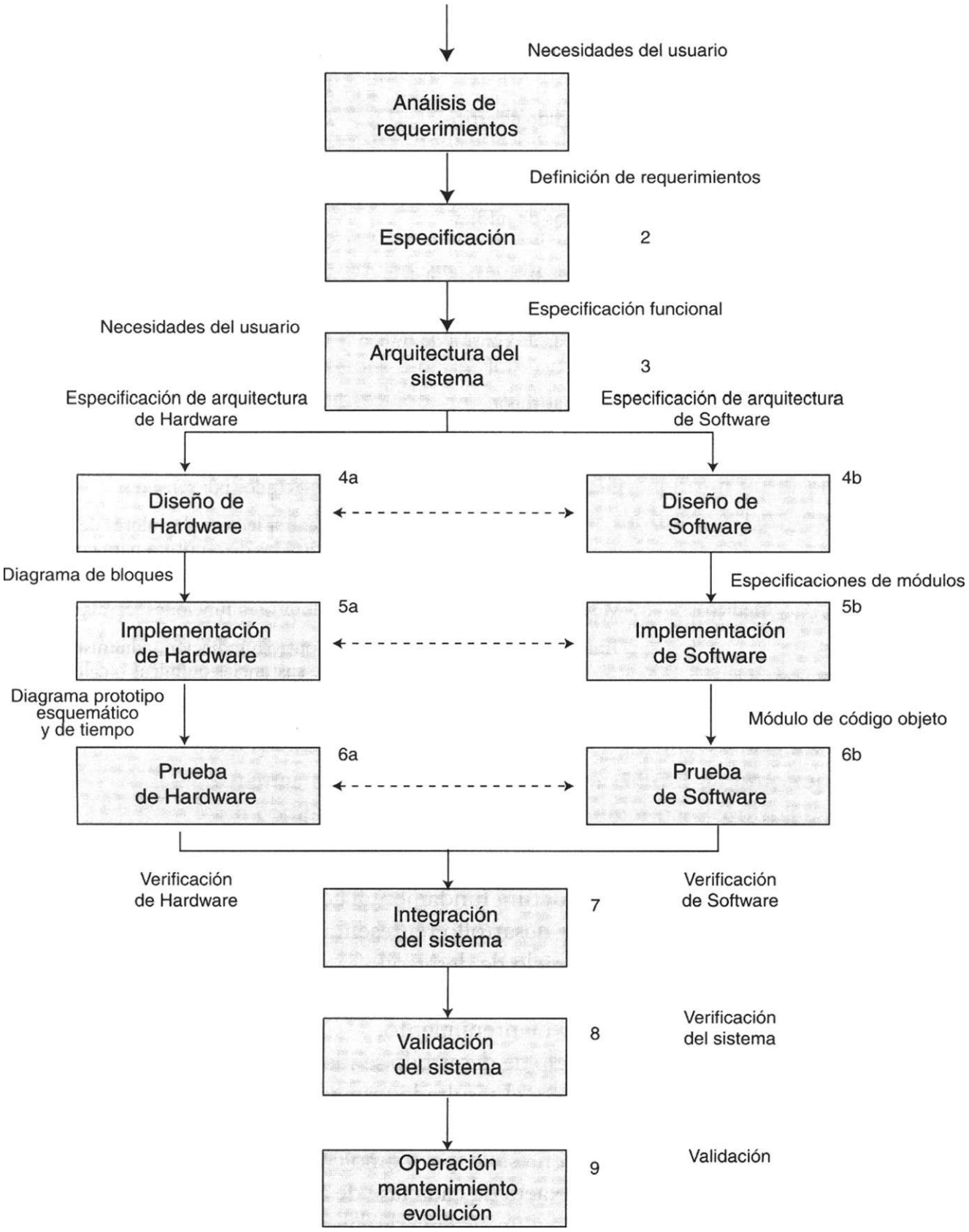


Figura 8.1 Ciclo de desarrollo de un sistema integrado.[2]

[2] Tomado de Short L.K.(1998).

El diagrama de la figura 8.1 ilustra las fases que ocurren por separado o en paralelo, pero sólo una a la vez y en secuencia. En realidad, el desarrollo de la información, los problemas encontrados y las decisiones tomadas durante alguna fase pueden causar que la fase interactúe con fases anteriores o posteriores, con lo que es posible la reactivación de las etapas previas. El resultado del proceso es iterativo y secuencial. A continuación describimos las fases del proceso.

Análisis de requerimientos

Este análisis representa un examen a detalle de las necesidades del usuario final; es decir, el problema por resolver. En esta etapa interviene el usuario final o cliente y sus necesidades, ya que éstas se estudian a fin de determinar qué quiere el usuario o qué necesita que haga el sistema. Estos requerimientos se definen a partir del punto de vista de este último y sin referencia alguna para posibles soluciones. Los resultados del análisis se registran en la definición de los requerimientos.

Especificaciones

Se basan en los requerimientos, funciones, operaciones y la interacción con el usuario. Definen al sistema y lo simulan para probarlo directamente; es decir, determinan la función del sistema, no cómo quedará.

Arquitectura del sistema

Este concepto se refiere a las especificaciones de software y hardware. En esta fase se define qué tipo se va a utilizar y cada elemento se describe por separado. Todo esto permite equilibrar el software y el hardware, aspecto fundamental para la ejecución y el costo del sistema. Esta etapa determina qué parte del sistema se implementa primero en el software y cuál en el hardware.

Diseño del hardware

Durante este paso se definen las funciones y la interfaz de entrada y salida; las primeras en forma general y la segunda de manera específica. Este diseño se puede establecer a gusto del usuario o siguiendo uno de los estándares aceptados por la industria. Esta fase incluye nombre de señales, funciones y características así como diagramas de bloques del sistema.

Implementación del hardware

En esta etapa se eligen los dispositivos (microprocesador, memoria, periférico, etc.) que van a integrar el sistema; también se lleva a cabo un análisis de

tiempo así como diagramas esquemáticos y diagramas de tiempo. Todo lo anterior permite la construcción de un prototipo.

Pruebas de hardware

Esta etapa consiste en realizar pruebas individuales a los dispositivos para determinar sus especificaciones y el tiempo que precisan a fin de ejecutar un proceso.

Diseño del software

En esta etapa se diseña la ejecución del flujo de datos y se implementan las funciones y procedimientos de cada módulo para que interactúen. Lo anterior permite jerarquizar el funcionamiento de los módulos que componen al sistema.

Implementación del software

Consiste en definir los algoritmos a detalle y las estructuras de datos que se van a desarrollar; además, el software se diseña de modo que pueda correr en otras aplicaciones —aunque siempre se diseña para una aplicación específica—. La programación del sistema se realiza mediante módulos independientes que más tarde se ensamblan o compilan. Todos los errores detectados se corrigen en este momento.

Pruebas del software

Los módulos programados en la etapa de implementación del software se prueban en forma individual mediante la simulación. Después de esto se conectan para tener el programa completo y probar cada estado.

Sistema de integración

Si las fases de software y hardware se ejecutaron por separado y todo funcionó, es el momento de integrar y probar el sistema de software y el prototipo del hardware. En este momento se detectan y corrigen los problemas encontrados en la interconexión o en las interfaces.

Validación del sistema

Esta fase permite saber si el sistema funciona, si hay errores —los cuales habrá que corregir— y realizar modificaciones solicitadas por el cliente.

Operación, mantenimiento y evolución

Es la fase final y en ella se implanta el sistema. Cuando éste ya se encuentra en servicio, se puede modificar para satisfacer nuevas necesidades e incrementar su calidad. En esta etapa se puede dar soporte al usuario o corregir algún error posterior.

8.1.2 Clasificación de los sistemas embebidos

La IEEE usa la clasificación que veremos a continuación.

- Microprocesadores individuales. Se pueden hallar en dispositivos pequeños como sensores de temperatura, detectores de gas y humo, circuitos de interrupción de corriente, etc. Ahora bien, conviene señalar que su manejo ofrece cierta dificultad.
- Líneas de producción pequeñas. Se pueden encontrar en controladores de flujo, amplificadores de señal, sensores de posición, accionadores de válvulas, etc. Su operación interna depende de un reloj.
- Líneas de producción con funciones de tiempo. Son los dispositivos de distribución eléctrica como controles, centrales telefónicas, sistemas de monitoreo y adquisición de datos, diagnóstico y sistemas de control en tiempo real, entre otros. Pueden ser elementos locales en sistemas más grandes, ya que proporcionan información de sus sensores; por ejemplo, en los sistemas de computación usados en la manufactura o en el control de procesos. Asimismo, pueden formar parte de una PC y comprender algunas bases de datos (como hechos). En estos casos la computadora se conecta a la planta o a la maquinaria para llevar el control de éstas. Los sistemas de computación se usan para controlar y monitorear el proceso. A menudo este tipo de sistemas abarca otros sistemas integrados que trabajan en conjunto.

8.1.3 Factores que se deben considerar para el diseño de un sistema embebido

Como puede advertirse, en el diseño de sistemas embebidos hay que considerar diferentes factores, de los cuales mencionaremos los más importantes a continuación:

- Costos de sistema que incluyen entrenamiento, desarrollo y pruebas.
- Costos de material que comprenden partes y refacciones.
- Costos de manufactura que dependen del número y complejidad de los componentes.
- Costos de mantenimiento que incluyen revisiones para arreglar errores y analizar si es posible aumentar o escalar el proyecto.

- La memoria de sólo lectura (ROM) debe tener el tamaño necesario para soportar instrucciones y arreglos de datos.
- La dimensión de la memoria de acceso rápido (RAM) debe ser suficiente para soportar variables globales y locales.
- La memoria EEPROM tiene que aceptar arreglos de consultas que están en un arreglo configurable.
- Es deseable que tenga velocidad para ejecutar el software en tiempo real.
- El ancho de banda de los dispositivos de entrada/salida afecta la rapidez de tránsito de datos en la entrada/salida.
- El tamaño de los datos de 8, 16 y 32 bits debe coincidir con los datos que se van a procesar.
- Las operaciones numéricas como multiplicaciones, divisiones, signo y punto flotante.
- Las funciones especiales como números complejos, lógica difusa, multiplicación/acumulador.
- Suficientes puertos paralelos para manejar todas las señales digitales de entrada/salida.
- Suficientes puertos seriales para aceptar la interfaz con otra computadora o dispositivo de entrada/salida.
- Funciones de tiempo para generar señales, mediciones de frecuencia y periodo de mediciones.
- Tamaño del circuito.
- Fuentes disponibles.
- Disposición de lenguajes de alto nivel, compiladores y simuladores.
- Requerimientos de poder, ya que muchos sistemas operan con pilas.

Además, la mayoría de los sistemas embebidos incluye componentes que miden y controlan parámetros del mundo real como posición, velocidad, temperatura y voltaje. Debido a esto el diseño siempre incluye dispositivos como amplificadores, filtros y convertidores digitales.

8.1.4 Diseño de microprocesadores VHDL

Como vimos antes, los sistemas embebidos tienen una infinidad de aplicaciones. Estos dispositivos realizan operaciones y controlan otros dispositivos, para lo cual reciben información y dan órdenes a fin de que los demás elementos trabajen.

El diseño de los microprocesadores se facilita gracias a la diversidad de herramientas que hay en la actualidad. Según vimos en el capítulo 1, el lenguaje VHDL es una herramienta muy poderosa para el diseño, pues además de su flexibilidad permite ahorrar en tiempo y costos.

8.2 Diseño de un microprocesador

En la figura 8.2 se muestra el diagrama a bloques del microprocesador que vamos a diseñar. Cabe mencionar que entre sus principales características se encuentran un bus de direcciones de 8 bits(A0-A7), un bus de datos de 4 bits (D0-D3), tres líneas de interrupción reset, IRQ1 e IRQ2, una línea de validación de direcciones (VMA), una línea para una señal de reloj, una línea de lectura/escritura y dos líneas para la alimentación.

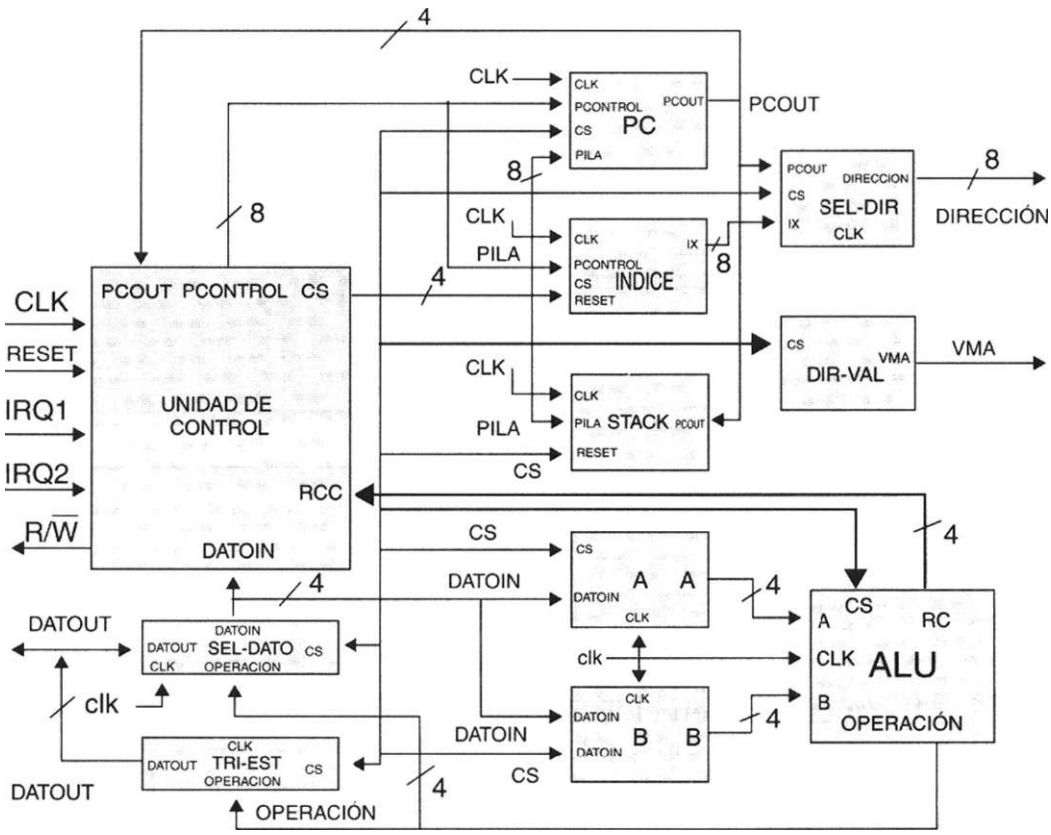


Figura 8.2 Diagrama a bloques del microprocesador.

Como puede observarse en la figura 8.2, el microprocesador se compone básicamente de 11 bloques, cada uno con una función específica. En lo que resta del capítulo se analizará cada bloque de manera detallada.

Este tipo de microprocesador es de clase Von Newman; es decir, los datos e instrucciones se encuentran en una memoria externa y el microprocesador por sí solo no trabajaría, ya que requiere ciertos circuitos auxiliares para su operación.

Contador de programa (PC)

El contador de programa (PC) es un registro de 8 bits que contiene la dirección del siguiente registro donde se encuentra la dirección (*nibble*) de la instrucción que se buscará en la memoria para su ejecución. El hecho que el contador de programa sea de 8 bits genera un total de 28 direcciones diferentes, es decir, 256 direcciones únicas, a las cuales el microprocesador debe habilitar cada una de ellas. El bus de direcciones del microprocesador es unidireccional y sólo permite salidas del microprocesador hacia los diferentes dispositivos. Cada vez que se ejecuta una instrucción, el contador de programa se incrementa. Su diagrama se muestra en la figura 8.3.

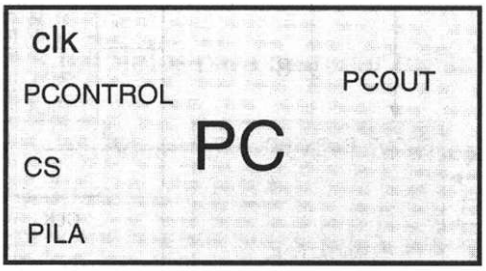


Figura 8.3 Contador de programa.

El código en VHDL que describe la función del contador de programa se encuentra en el listado 8.1.

```
library ieee;
use ieee.std_logic_1164.all ;
use work.std_arith.all ;
entity pe is port  (
            elk:  in std_logic;
       pcontrol:  in std_logic_vector  (7 downto 0);
             cs:  in std_logic_vector(4 downto 0);
          reset:  in std_logic;
           pila:  in std_logic_vector  (7 downto 0) ;
          pcout:  inout std_logic_vector  (7 downto 0)
end;
architecture arq pc of pc is
begin
```

Continúa

```
        process(elk, cs, reset)
        begin
            if reset = "1" then
             pc <= "00000000";
            else
    if (elk'event and elk = '1') then
              case cs is
                  when "11110" => pcout <= pcontrol;
                  when "11111" => pcout <= pcout + 1;
                  when "11101" => pcout <= pila;
                  when others => null ;
                  end case;
               end if;
           end if ;
        end process;
    end ;
```

Listado 8.1 Programación del contador de programa.

Según la descripción anterior en VHDL, el contador del programa tiene tres variables que lo hacen trabajar: es, *clk y reset.*

La variable *reset* sirve para inicializarlo en la dirección "00000000", misma en que el microprocesador ejecuta la primera instrucción.

La variable *es* indica la función que se debe realizar. Cuando *es* es igual a "11110" la variable *pcout* es igual a la variable *pcontrol* —esta última señal proviene directamente de la unidad de control—. Además, puede ser igual a la dirección de inicio, a una nueva dirección cuando se dio antes una instrucción de salto y a las direcciones de inicio de las interrupciones *IRQ1* y *1RQ2.*

Cuando la variable *es* del contador del programa es igual a "11111" el valor de *pcout* se incrementa una unidad. Esto ocurre cuando se está ejecutando la instrucción presente y requiere un dato o la siguiente instrucción. Cuando es es igual a "11101", *pcout* toma el valor que está almacenado en la pila. Esto sucede cuando el microprocesador atendió una petición de interrupción *(IRQ1* e *IRQ2)* y necesita regresar a la dirección donde estaba en un inicio. Cualquier otra combinación en es tiene un efecto nulo en el contador de programa.

La señal de *clk* indica al microprocesador cuándo ejecutar cada una de las acciones referidas. En todos los bloques del microprocesador, la señal *clk* tiene efecto cuando pasa de un bajo a un alto.

Registro de índice

El registro de índice (IX) es un registro de 8 bits que se usa para almacenar una dirección donde es posible escribir datos almacenados en el acumulador A. Este registro se puede incrementar e inicializar a una dirección que el usuario especifique.

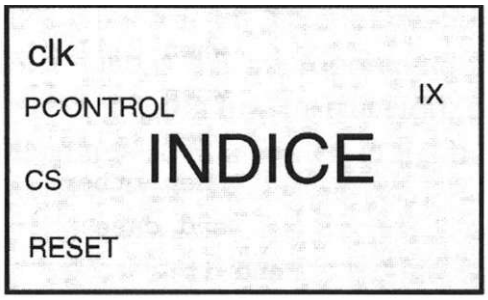


Figura 8.4 Registro de índice.

El código en VHDL registro de índice se encuentra en el listado 8.2.

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity indice is port (
            elk: in std_logic;
       pcontrol: in std_logic_vector (7 downto 0);
             cs: in std_logic_vector(4 downto 0);
          reset: in std_logic;
             ix: inout std_logic_vector (7 downto 0) ) ;
end;
architecture arq_indice of indice is
begin
      process(elk, cs, reset)
      begin
       if reset = "1" then
          ix <= "00000000";
      else
if (elk'event and elk = '1') then
                  case cs is
                  when "11010" => ix <= pcontrol;
                  when "11011" => ix <= ix + 1;
                  when others => null;
                  end case;
            end if;
        end if;
      end process;
end;
```

Listado 8.2 Programación del registro de índice.

Las señales es, *clk* y *reset* afectan el registro de índice. La señal *clk* tiene el mismo efecto que en el contador de programa.

Cuando la señal de *reset* es igual a "1", coloca el contador de índice (IX) con la dirección inicial "00000000". Cuando la señal de es es igual a "11010" el contador de índice es igual a *pcontrol.* En este caso, el usuario determina el valor de *pcontrol* en su programa. Cuando es es igual a "11011" el valor de IX se incrementa una unidad. Cualquier otra combinación de *es* tiene un efecto nulo en el registro de índice.

Registro de pila (STACK)

La pila *(stack)* es un registro de 8 bits que se usa para almacenar la última dirección del contador de programa cuando ocurre una solicitud de interrupción *(IRQ1* y *IRQ2):* Est se ve en la figura 8.4.

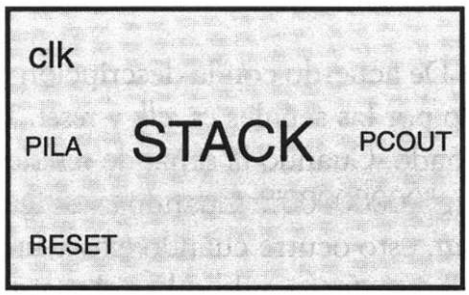


Figura 8.4 Registro de pila *(stack).*

Su código en VHDL se muestra en el listado 8.3.

```
library ieee;
use  ieee.std_logic_1164.all;
use work.std_arith.all;
entity stack is port  (
                      clk: in std_logic;
                    pcout: in std_logic_vector  (7 downto 0);
                     pila: inout std_logic_vector  (7 downto 0);
                    reset: in std_logic;



                       es: in std_logic_vector  (4 downto 0));
end;
architecture arq_stack of  stack is
begin
```

Continúa

```
      process (elk, cs, reset)
      variable q: std_logic_vector (7 downto 0);
      begin
        if reset = '1' then
            pila <="00000000";
        else
             if (elk'event and elk = '1') then
               case cs is
               when "11100" => pila <= pcout;
               when others => null;
               end case;
         end if;
       end if;
      end process;
end;
```

Listado 8.3 Programación de la pila.

De acuerdo con la descripción VHDL del registro de pila, éste se ve afectado por las señales cs, *clk* y *reset.* La señal *clk* funciona como ya se ha mencionado. Cuando la señal de *reset* es igual a "1", inicializa la pila con la dirección "00000000". Cuando *cs* es igual a "11100", el valor de la pila es igual al *pcout,* esto ocurre cuando existe una interrupción. Cualquier otro valor en *cs* tiene un efecto nulo sobre el registro de pila.

Acumulador A y B

El microprocesador contiene dos acumuladores (A y B) de 4 bits cada uno, que se usan como registros temporales. Contiene los operandos y el resultado de las operaciones efectuadas en la unidad lógica aritmética.

El código en VHDL del acumulador A se presenta en el listado 8.4.

```
library ieee;
use ieee.std_logic_1164.all ;
entity rega5 is port (
                        cs:in std_logic_vector (4 downto 0);
                       clk: in std_logic;
                datoin:in std_logic_vector (3 downto 0);
               reset: in std_logic;
                     a:inout std_logic_vector (3 downto 0));
end;
architecture impedancia of rega5 is
begin
```

Continúa

```
process(cs, clk, reset)
variable ares:std_logic_vector (3 downto 0);
    begin
      if reset = "1" then
            a <= "0000";
      else
              if (elk'event and clk = '1') then
            case cs is
                when "10001" =>
                a <= datoin;
                when "11100" =>
                ares := a;
                when "11101" =>
                a <= ares;
                when others =>
                null ;
            end case;
          end if ;
      end if;
  end process;
end;
```

Listado 8.4 Programación del acumulador A.

Las señales *clk, reset* y *es* modifican el valor del acumulador A. El contenido de éste depende del usuario y del resultado de las operaciones efectuadas en la unidad aritmética y lógica. Cuando el *reset* tiene un valor de "1", el contenido del acumulador es "0000". La variable interna *ares* desempeña una función importante cuando existe una interrupción, ya que guarda el contenido del acumulador, mismo que devuelve cuando el microprocesador regresa de dicha interrupción para que la ejecución del programa principal no sufra alteraciones; en la descripción VHDL se ve con claridad qué combinaciones de *es* realizan esta función.

Además, la combinación de *es* igual a "10001" permite almacenar un dato en el acumulador para su procesamiento.

En el listado 8.5 se muestra el código en VHDL del acumulador B.

```
library ieee;
use ieee.std_logic_1164.all;
entity regb5 is port (
                    cs:in std_logic_vector (4 downto 0);
            clk,reset:in std_logic;
               datoin:in std_logic_vector (3 downto 0);
                    b: inout std_logic_vector (3 downto 0));
end ;
```

Continúa

```
architecture impedancia of regb5 is
begin
        process(cs, clk, reset)
        variable bres:std_logic_vector (3 downto 0);
     begin
          if reset = "1" then
             b <= "0000";
              else
              if (clk'event and clk = '1') then
              case cs is
when "10010" =>
                        b <= datoin;
                        when "11100" =>
                        bres := b;
                        when "11101" =>
                        b <= bres;
                        when others =>
                        null;
                    end case;
          end if;
        end if;
              end process;
end;
```

Listado 8.5 Programación del acumulador B.

El acumulador B realiza una función similar al acumulador A. La única variación está en el contenido, que depende exclusivamente del valor que asigna el usuario. Para ello la variable *cs* debe ser igual a "10010. La variable *bres* tiene la misma función que la variable *ares* en el acumulador A.

Registro Sel-Dato

Es un registro auxiliar que permite introducir datos al bus de datos interno del microprocesador. Estos datos tienen como destino los acumuladores A o B y la unidad de control.

El código en VHDL de este registro se muestra en el listado 8.6.

```
library ieee;
use ieee.std_logic_1164.all ;
entity sel_dato is port  (
                        clk:in std_logic;
                   datout:in std_logic_vector  (3 downto 0);
                   datoin:inout std_logic_vector  (3 downto 0);
                operacion:in std_logic_vector  (3 downto 0);
                       cs:in std_logic_vector  (4 downto 0) ) ;
end;
architecture arq_sel_dato of sel_dato is
begin
       process(cs,clk}
         begin
               if  (clk'event and clk =  '1')  then
               case cs is
                 when  "10110"  => datoin <= datout;
                 when  "10101"  => datoin <= operacion;
                 when others => null;
                 end case ;
               end if ;
       end process ;
end ;
```

Listado 8.6 Programación del registro Sel-dato.

Como se observa, en el código del Sel-Dato la variable *reset* no tiene efecto alguno; en cambio, la señal *clk* tiene la función ya mencionada.

La señal cs determina el tipo de dato que se va a introducir. Este dato puede ser externo, de una memoria, y se presenta cuando cs es igual a "10110", o interno, como resultado de una operación en la unidad lógica aritmética. Esto último sucede cuando cs es igual a "10101". En síntesis, el registro Sel-Dato no es sino un multiplexor de dos entradas con una salida.

Registro Tri-Est

Es un registro que permite sacar datos del microprocesador, los que pueden ser el resultado de una operación en la unidad aritmética y lógica. Cuando no está en uso este registro, sus salidas tienen un estado de alta impedancia.

En el listado 8.7 se muestra el código VHDL correspondiente.

```
library ieee;
use ieee.std_logic_1164.all;
entity tri_esr is port (
                  cs:in std_logic_vector (4 downto 0);
                 clk: in std_logic;
           operacion:in std_logic_vector (3 downto 0);
              datout:inout std_logic_vector (3 downto 0));
end;
architecture arq_tri of tri_esr is
begin
      process(cs, clk)
      begin
            if (clk'event and clk = '1') then
            case cs is
               when "11000" => datout <= operacion;
               when others => datout <= "ZZZZ";
               end case;
           end if;
      end process;
end;
```

Listado 8.7 Programación del registro Tri-Est.

Como puede observarse, *reset* tampoco afecta al registro Tri-Est y según su descripción, se trata de un buffer unidireccional con estado de alta impedancia. En conjunto, los registros Sel-Dato y Tri-Est dan al bus de datos del microprocesador la característica de ser bidireccional.

Registro Sel'Dir

El registro Sel-Dir permite que las direcciones generadas por el contador de programa o las del registro índice salgan del microprocesador con objeto de leer o escribir un dato, según el valor de la señal R/**W.**

El código que describe al registro Sel-Dir se encuentra en el listado 8.8.

```
library ieee;
use  ieee.std_logic_1164.all;
entity sel_dir is port  (
                pcout:in std_logic_vector  (7  downto  0);
                  clk:in  std_logic;
                   ix:in std_logic_vector  (7  downto  0);
            dirección:out std_logic_vector  (7  downto  0) ;
                   cs:in std_logic_vector  (4  downto  0));
end  sel_dir;
architecture arq_sel_dir of  sel_dir  is
begin
       process(es,  clk)
               begin
               if  (clk'event and clk =  '1')  then
               case  cs  is
               when  "11000"  => dirección <= ix;



               when others   => dirección <= pcout;
               end case;
               end if;
            end process;
end;
```

Listado 8.8 Programación del registro Sel-Dir.

La mayoría de las veces, el registro Sel-Dir selecciona la dirección generada por el contador de programa, debido a que el microprocesador siempre está leyendo datos o instrucciones y nada más escoge la dirección del registro índice cuando va a escribir un dato en la dirección que éste señala. Aunque en el código de VHDL de Sel-Dir no existe la señal de *reset,* cuando ésta es igual a "1" el valor de dirección es igual a "00000000". Al igual que en el registro Sel-Dato, Sel-Dir no es más que un multiplexor de dos entradas y una salida.

Registro Dir-Val

La importancia de este registro reside en que su salida indica a los dispositivos periféricos que la dirección presente en el registro Sel-Dir es válida. En operación normal, la señal Dir-Val (VMA) se debe utilizar a fin de habilitar dispositivos periféricos como memorias, decodificadores o algún otro dispositivo de entrada/salida. Esta señal no es de tres estados y cuando tiene un valor de "1" indica que la dirección es válida; en caso contrario es una dirección no válida. Esto es así porque durante algunas operaciones internas el microprocesador coloca una dirección en el bus que no está usando en ese momento.

Su código en VHDL se muestra en el listado 8.9.

```
library ieee;
use ieee.std_logic_1164.all ;
entity dir_val is port (
                vma: out s td_logic;
                 cs:in std_logic_vector (4 downto 0) ) ;
end;
architecture arq_dir_val of dir_val is
begin
      process(cs)
            begin
            vma <= (not (cs(3)) and cs(2) and cs(l) and not (cs(0)))
or (cs(3) and not (cs{2)) and not (cs(l)) and not (cs(0)));
          end process;
end;
```

Listado 8.9 Código del registro Dir-Val.

Como puede observarse, el registro Dir-Val es un circuito combinatorio con una salida única (VMA) y su valor es igual a "1" cuando se efectúa una operación de lectura o escritura. El valor de VMA depende en todo momento de es.

Unidad aritmética y lógica (ALU)

La unidad aritmética y lógica es una función multioperación digital de lógica combinacional. Puede realizar un conjunto de operaciones aritméticas básicas y otro de operaciones lógicas.

El código en VHDL de la unidad aritmética y lógica se encuentra en el listado 8.10.

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity alu4 is port (
            clk:in std_logic;
            a,b:in std_logic_vector (3 downto 0);
                cs:in std_logic_vector (4 downto 0);
         operacion:inout std_logic_vector (3 downto 0);
                re:inout std_logic_vector (3 downto 0));
end alu4;
architecture sumar of alu4 is
signal el:std_logic_vector (1 downto 0);
begin
     process(clk)
     variable Cout: std_logic;
     begin
     if (clk'event and clk = '1') then
--Selecciona una operación,
          case cs is
          when "00001" =>
          operacion <= (a + b) ;
          cl(0)<=(a(1) and b(1)) or ((a(0) and b(0)) ard (a(1) xorb(1)));
          cl(1)<= (a(2) and b(2)) or (cl(0) and (a(2) xorb(2)));
          Cout:=(a(3) andb(3)) or (cl(1) and (a(3) xor b(3)));
```

Continúa

```
                    when "00010" =>
                    operacion <= (a - b) ;
                    if (a >= b) then
                    Cout := '1';
                    C1(0)<= '1' ;
                    else
                    Cout := ' 0 ' ;
                    cl(0)<= •0';
                    end if;

                    when "00011" =>
                    operacion <= (a and b) ;
                    Cout := 10' ;
                    cl(0)<= '0';

                    when "00100" =>
                    operacion <= (a or b) ;
                    Cout := ' 0 ' ;
                    cl(0)<= 10';

                    when "00101" =>
                    operacion <= (not a) ;
                    Cout := '0';
                    cl (0)<= ' 0 ' ;
                    when "00110" =>
                    operacion <= (a xor b) ;
                    Cout := '0';
                    cl (0)<= '0';

                    when "00111" =>
                    operacion <= (a and "1111");
                    Cout := Cout;
                    cl(0) <= cl(0);

                    when others =>
                    null ;
                end case;
                end if;
                    rc(3)<= Cout xor cl(l); -- Sobreflujo
                    rc(2)<= not(operacion(3) or operacion(2) or
operacion(1) or operacion(0));
                    -- Cero
                    rc(l)<= operacion(3); —Signo
                    rc(0)<= Cout; --Acarreo
            end process;
    end;
```

Listado 8.10 Descripción de la unidad aritmética y lógica.

La unidad aritmética y lógica del microprocesador se compone de dos entradas de 4 bits cada una (contenido de los acumuladores A y B) y una salida de 4 bits (operación) con acarreo. Puede realizar siete operaciones: dos aritméticas (suma y resta), cinco lógicas (AND, OR, NOT, OR EXCLUSIVA) y enmascarar el contenido del acumulador A.

El diseño de esta unidad aritmética y lógica supone la generación del registro de códigos de condición (rcc). Este registro es de 4 bits e indica cómo es el resultado de una operación en la unidad aritmética y lógica; estos cuatro bits indican si el resultado es negativo (N), igual a cero (Z), si tiene sobreflujo (V) y si hay acarreo (C).

- Acarreo (C). Este bit (bit 0) del registro de códigos de condición se coloca en "1" si después de la ejecución de ciertas instrucciones hay un acarreo del bit más significativo de la operación que se está ejecutando; de otra manera se coloca en "0".
- Sobreflujo (V). Este bit (bit 3) del registro de códigos de condición se pone en "1" cuando un sobreflujo en complemento a 2 resulta de una operación aritmética; se coloca en "0" si el sobreflujo no ocurre en ese tiempo. Por lo general hay sobreflujo cuando la última operación resulta ser un número mayor que ± 7 de un registro de 4 bits.
- Cero (Z). Este bit (bit 2) del registro de códigos de condición se coloca en "1" si el resultado de la operación lógica o aritmética es cero; de otra manera se pone en "0".
- Negativo (N). Este bit (bit 1) del registro de códigos de condición se coloca en "1" si el bit 4 del resultado de una operación lógica o aritmética es igual a "1", de lo contrario se coloca en "0".

Aunque son pocas las operaciones que se pueden realizar con esta unidad aritmética y lógica, no hay que olvidar que al combinar estas operaciones básicas el número de éstas crece de manera importante y así es posible realizar decrementos, incrementos, corrimientos a la izquierda o derecha, corrimientos aritméticos, funciones NAND, ÑOR, etcétera.

En lo que se refiere a las señales que controlan la unidad aritmética y lógica, el valor de *es* determina qué operación se va a llevar a cabo en dicha unidad y la señal de *clk* marca en qué tiempo se realizará.

Unidad de control

La unidad de control (CU) sincroniza cada una de las acciones realizadas por el microprocesador; asimismo, determina en qué tiempo, hacia dónde se mandan los datos o de dónde vienen, decodifica cada instrucción y revisa que se ejecute.

El código de la unidad de control se presenta en el listado 8.11.

```
library ieee;
use ieee.std_logic_1164.all ;
use work.std_arith.all;
entity control is port(
            reset,clk: in std_logic;
                  irq: in std_logic_vector (1 downto 0);
                   rw: inout std_logic;
              datoin: in std_logic_vector (3 downto 0);
             pcontrol: inout std_logic_vector (7 downto 0);
                   re: in std_logic_vector (3 downto 0);



                 pcout: in std_logic_vector (7 downto 0);
                    cs: inout std_logic_vector (4 downto 0));
end control;
architecture arq_control of control is
type estados is (d0, di, d2, d3, d4, d5, d6, d7, d8, d9, dlO,
dll, dl2, dl3, dl4, dl5, dl6, dl7);
signal edo_presente, edo_futuro: estados;
signal f: std_logic_vector (7 downto 0);
begin
     procesol: process (edo_presente, irq, reset)
          begin
              if reset = '1' then
                  edo_futuro <= d0;
                  pcontrol <= "11111111";
                       rw <= '0';
```

Continúa

```
        CS <= "11110";
    else
case edo_presente is
        when d0 =>
        if irq = "10" then
           es <= "11100";
           edo futuro <= di;
        elsif irq = "01" then
           es <= "11100";
           edo_futuro <= di;
        else
           edo_futuro <= d3;
           es <= "11111";
        end if;
        when di =>

        if irq = "10" then
           pcontrol <= "01000000";
           edo_futuro <= di;
        elsif irq = "01" then
           pcontrol <= "10000000';
           edo_futuro <= di;
        else
           es <= "11110";
           edo_futuro <= d3;
        end if;

        when d2 =>
           es <= es;
           edo_futuro <= d5;

        when d3 =>
              es <= "10110";
              rw <= '1';
            edo_futuro <= d4;

        when d4 =>
              rw <= ' 0';
        if datoin = "0000" then
              es <= "00001";
         edo_futuro <= d2;
        elsif datoin = "0001" then
              es <= "00010";
         edo_futuro <= d5;
```

Continúa

```
elsif datoin = "0010" then

      cs <= "00011";
  edo_futuro <= d5;

elsif datoin = "0011" then
      cs <= "00100";
  edo_futuro <= d5;

elsif datoin = "0100" then
      cs <= "00101";
  edo_futuro <= d5;

elsif datoin = "0101" then
      cs <= "00110";
  edo_futuro <= d5;

elsif datoin = "0110" then
      cs <= "11111";
  edo_futuro <= d6;

elsif datoin = "0111" then
      cs <= "11111";
  edo_futuro <= d7;

elsif datoin = "1000" then
      cs <= "11111";
  edo_futuro <= d8;

elsif datoin = "1001" then
      cs <= "00111";

  edo_futuro <= d9;

elsif datoin = "1010" then
      cs <= "11011";
  edo_futuro <= d0;

elsif datoin = "1011" then
      cs <= "11111";
```

Continúa

```
    if re(0) = '1' then
   edo_futuro <= dlO;
      else
   edo_futuro <= dO;
      end if;

elsif datoin = "1100" then
         es <= "11111";
    if re(2) = '1' then
   edo_futuro <= dlO;
      else
   edo_futuro <= dO;
      end if;

elsif datoin = "1101" then
      es <= "11111";
    if re(1) = '1' then
   edo_futuro <= dlO;
    else

   edo_futuro <= dO;
      end if;
elsif datoin = "1110" then


      es <= "11111";
    if re(3) = '1' then
   edo_futuro <= dlO;

    else
   edo_futuro <= dO;
      end if;

    else
      es <= "11101";
       edo_futuro <= dO;
      end if;

      when d5 =>
           es    "10101";
       edo_futuro <= dll;

      when d6 =>
           es <= "10110";
           rv; <= '1';
       edo_futuro <= dl2;
```

Continúa

```
when d7 =>
      cs <= "10110";
      rw <= '1';
 edo_futuro <= dll;

when d8 =>
      cs <= "10110";

      rw <= '1';
 edo_futuro <= dl5;


when d9 =>
      cs <= "11000";
 edo_futuro <= dO;

when dlO =>
      cs <= "10110";

      rw <= '1';
 edo_futuro <= dl6;

when dll =>
      CS <= "10001";
      rw <= '0';
 edo_futuro <= dO;

when di2 =>
      CS <= "11111";
      rw <= '0';
      pcontrol(O) <= datoin(O);
      pcontrol(1) <= datoin(l);
      pcontrol(2) <= datoin(2);
      pcontrol(3) <= datoin(3);
      pcontrol(4) <= pcontrol(4);
      pcontrol(5) <= pcontrol(5);
      pcontrol(6) <= pcontrol(6);
      pcontrol(7) <= pcontrol(7);
```

Continúa

```
 edo_futuro <= dl3;

when dl3 =>
      es <= "10110";
      rw <= '1';
 edo_futuro <= dl4;

when dl4 =>
      rw <= '0';
      es <= "11010";
      pcontrol(0) <= pcontrol(0);
      pcontrol(l) <= pcontrol(l);
      pcontrol(2) <= pcontrol(2);
      pcontrol(3) <= pcontrol(3);
      pcontrol(4) <= datoin(0);
      pcontrol(5) <= datoin(l);
      pcontrol(6) <= datoin(2);
      pcontrol(7) <= datoin(3);
 edo_futuro <= dO;

when di5 =>
      es <= "10010";
      rw <= '0';
 edo_futuro <= dO;

when di6 =>
      es <= "10000";
      f(0)<= datoin(0);
      f(1)<= datoin(1);
      f(2)<= datoin(2);
      f(3)<= '0';
      f(4)<="0';
      f(5)<= '0';
      f(6)<= '0';
      f(7)<= '0';
    if datoin(3) = '0' then
  pcontrol <= pcout + f;
  edo_futuro <= dl7;
  else
  pcontrol <= pcout - f;
  edo_futuro <= dl7;
  end if;

when di7 =>
```

Continúa

```
                                    cs <= "11110";
                                edo_futuro <= d0;
                            end case;
                        end if;
end process procesol;

proceso2: process(clk, reset)
begin
      if (clk'event and clk = '1') then
        if reset = '0' then
          edo_presente <= edo_futuro;
        else
        null;
       end if;
     end if;
     end process proceso2;
end;
```

Listado 8.11 Descripción de la unidad de control.

De acuerdo con el código en VHDL, la unidad de control es una máquina de estados finitos. En ella se utiliza un proceso combinacional para describir la función del próximo estado, las asignaciones de salida y un proceso secuencial para describir las asignaciones sobre el registro de estados en la transición activa de la señal de reloj.

La unidad de control genera la señal cs para cada bloque. Esta señal les marca la función que van a realizar y en qué tiempo deben efectuarla; de igual forma, genera la señal de lectura/escritura, la señal de *pcontrol* que va directamente al registro del contador de programa que genera las direcciones. Las señales *clk, reset, IRQ1, IRQ2* y rcc son señales de entrada; la última resulta de las operaciones realizadas en la unidad aritmética y lógica e indica a la unidad de control cómo es el resultado.

Al inicio del código en VHDL, las interrupciones tienen una función preponderante porque indican al microprocesador dónde leer la primera instrucción.

La interrupción de mayor importancia es *reset,* pues inicializa al microprocesador empezando por la misma unidad de control, coloca al *pcontrol* con valor de "11111111" y le dice al contador de programa que se incremente. Después de *reset* puede haber dos interrupciones *IRQ1* e *IRQ2.* En caso de presentarse alguna de ellas el *pcontrol* se modifica, pero antes el microprocesador guarda el resultado de los dos acumuladores y de la última dirección (del contador de programa). Si se da una *IRQ1,* el *pcontrol* es igual a "10000000"; si se presenta una interrupción *IRQ2,* el valor de pcontrol es

igual a "01000000". Ambas direcciones indican dónde iniciar las rutinas de interrupción. Al terminar las rutinas, los acumuladores vuelven a su valor original, al igual que el valor del contador de programa. En caso de no existir alguna interrupción, el microprocesador genera la dirección "00000000" y la señal R/W se va a "1". Con esto el microprocesador lee el contenido de esta dirección, que es una instrucción, e inicia la secuencia de ejecución.

En la figura 8.5 se puede observar un diagrama que indica el flujo de la operación que sigue el microprocesador al momento de empezar a trabajar y el camino que toman las interrupciones.

La unidad de control está compuesta por 17 estados, cada uno con una función específica. Los estados "0" y "1" se encargan de las tres interrupciones que existen. Se llega al estado "0" después de un *reset* o luego de correr la última instrucción.

En el estado "0" se da la orden para incrementar el contador de programa (es = "11111"). Del estado "0" se va al estado "1", a menos que entre un *reset.* En éste la unidad de control comprueba si ha entrado una interrupción. En caso positivo cambia el contador de programa y pasa al estado "3", en caso contrario pasa de igual forma al estado "3". En este estado la unidad de control lee la instrucción. Para ello las señales r/W y VMA se van a "1", y la señal es es igual a "10110".

Este valor de es indica al registro Sel-Dato que deje pasar el dato externo al bus de datos interno del microprocesador *(datoin),* para que sea analizado en la unidad de control. De aquí se va directamente al estado "4", que es el más importante, ya que cumple las funciones de decodificador de instrucciones y determina el camino que se debe seguir en la ejecución de las instrucciones.

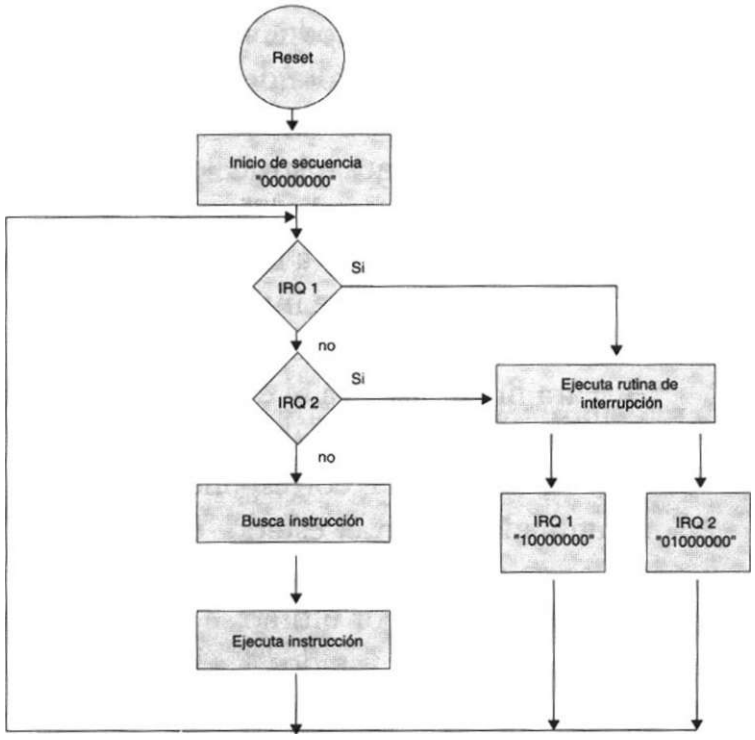


Figura 8.5 Diagrama de flujo de interrupciones.

En el estado "4" se tienen 16 combinaciones de la señal *datoin* y cada una representa una instrucción.

Las 16 instrucciones se dividen en las siguientes categorías:

Número de instrucción	Tipo de instrucción
7	Lógica-aritmética
4	Bifurcación
4	Carga en acumulador, índice y dirección
1	Retorno de interrupción

Más adelante se da una lista detallada de las 16 instrucciones que puede ejecutar el microprocesador.

En el estado "4" nada más se ejecutan las instrucciones lógico aritméticas y se indica a la ALU el tipo de función que se va a realizar. Para este tipo de instrucciones se pasa al estado "5", donde la unidad de control coloca el resultado de la operación en el bus de datos interno. A continuación se va al estado "11", donde el resultado se almacena en el acumulador A. Por último se pasa al estado "0" donde todo comienza de nuevo.

Para llegar al estado "6" es necesario que en el estado "4" el valor de *datoin* sea igual a "0110". Este dato corresponde a la instrucción de cargar una dirección en el registro índice. En el estado "6" la unidad de control da la orden para introducir un dato. De aquí se pasa al estado "12", donde el dato que se introdujo en el estado anterior corresponde a la parte baja de la dirección que se va a cargar en el registro índice. En el estado "13" se da la orden de introducir un segundo dato y en el estado "14" este dato pasa a formar la parte alta de la dirección y desde este estado se carga esta dirección ya formada en el registro índice. De aquí se va al estado "0" para ejecutar la siguiente instrucción.

Para llegar al estado "7", en el estado "4" el *datoin* debe ser igual a "0111". Este dato corresponde a la instrucción de cargar un dato en el acumulador A.

En el estado "7" se introduce el dato que se quiere cargar. De ahí se pasa al estado "11", donde la unidad de control coloca el dato en el acumulador A. Para finalizar se va al estado "0" a esperar la siguiente instrucción.

Para llegar al estado "8", en el estado "4" el valor de *datoin* debió ser igual a "1000". Este valor corresponde a la instrucción de cargar un dato en el acumulador B. En el estado "8" se introduce el dato que se va a cargar. De ahí se va al estado "15", donde la unidad de control coloca dicho dato en el acumulador B y por último se pasa al estado "0".

Cuando en el estado "4" el *datoin* es igual a "1001", esto significa que hay una instrucción de almacenar el dato del acumulador A en la dirección contenida en el registro índice. En este estado la unidad de control coloca el valor

del acumulador A en el bus de datos interno. Luego se pasa al estado "9", donde la dirección del índice se coloca en el registro Sel-Dir y el registro Tri-Est es habilitado para sacar el valor del acumulador del microprocesador. De aquí se salta al estado "0".

Las siguientes cuatro instrucciones son de bifurcación y todas pasan por los mismos estados. La única diferencia es la condición que tienen que comprobar para validar la condición de salto o no. Si en el estado "4" se da alguno de los siguientes valores de *datoin*: "1011", "1100", "1101" y "1110", se trata de la instrucción de salta si hay acarreo (C = 1), salta si es igual a cero (Z = 1), salta si es negativo (N — 1) y salta si hay sobreflujo (V = 0), respectivamente. En el mismo estado "4" se comprueba si la condición de salto es verdadera. Si es falsa se pasa al estado "0" y en caso contrario se va al estado "10", donde se introduce el dato que determina el valor del salto; de aquí se pasa al estado "16". Este salto puede ser hacia delante o hacia atrás y su valor máximo es ± 7 direcciones a partir de la dirección que contiene el dato que determina su longitud. El signo del salto se determina a partir del valor del tercer bit del dato: si es igual a "0" es positivo y si es igual a "1" es negativo.

A continuación se pasa al estado "17", donde la nueva dirección se coloca en el contador de programa. Para finalizar se va al estado "0".

Cuando en el estado "4" se tiene un valor de *datoin* igual a "1010", esto significa que hay una instrucción de incrementar el valor del registro índice en el mismo estado "4" —la unidad de control realiza esta operación—. De aquí se regresa al estado "0".

La última instrucción se utiliza cuando ha habido previamente una interrupción *(1RQI* e *IRQ2).* Si en el estado "4" el valor de *datoin* es igual a "1111", esto significa que la rutina de interrupción ha terminado y que el microprocesador tiene que regresar a las condiciones previas a la interrupción. Para ello la unidad de control saca la dirección almacenada en la pila, la coloca en el contador de programa y el valor original de los acumuladores se carga de nuevo. Enseguida se va al estado "0".

8.3 Diseño jerárquico

Como se mencionó en el capítulo 7, el diseño jerárquico es una herramienta de apoyo que permite la programación de extensos diseños mediante la integración de pequeños bloques, los cuales se pueden detallar y simular por separado con suma facilidad.

Una vez estudiado cada uno de los bloques que conforman al microprocesador, es necesario aclarar que de acuerdo con la programación en VHDL es necesario crear dos programas: uno que defina cada uno de estos bloques como componentes y otro —denominado de alto nivel— que encadena cada uno de estos componentes y los hace trabajar en conjunto.

8.3.1 Creación del archivo de componentes

El archivo que define cada uno de estos bloques como componentes aparece en el listado 8.12.

```
library ieee;
use ieee.std_logic_1164.all ;
package cmicro is

        component alu4 port  (
                        clk:in std_logic;
                        a,b:in std_logic_vector  (3 downto 0);
                         cs:in std_logic_vector  (4 downto 0);
                    operacion: inout std_logic_vector  (3 dcwnto 0);
                          re:inout std_logic_vector  (3 downto 0));
        end component;

        component rega5 port  (
                       cs:in std_logic_vector  (4 downto 0);
                    reset:in std_logic;
                      clk:in std_logic;
                   datoin:in std_logic_vector  (3 downto 0);
                      a:inout std_logic_vector  (3 downto 0));
        end component;

        component regb5 port  (
                    reset:in std_logic;
                       cs:in std_logic_vector  (4 downto 0);
                      clk:in std_logic;
                   datoin:in std_logic_vector  (3 downto 0);
                        b:inout std_logic_vector  (3 downto 0));


        end corrponent ;

        component sel_dato port  (
                      clk:in std_logic;
                   datout:in std_logic_vector  (3 downto 0);
                   datoin:inout std_logic_vectcr  (3 downto 0);
               operacion:in std_logic_vector  (3 downto 0);
                       cs:in std_logic_vector  (4 downto 0));

        end component;

        component tri_esr port  (
```

Continúa

```
                cs:in std_logic_vector (4 downto 0) ;
          operacion:in std_logic_vector (3 downto 0) ;

             datout:inout std_logic_vector (3 downto 0));
end component;

component pe port (
               clk: in std_logic;
             reset: in std_logic;
   pcontrol, pila: in std_logic_vector (7 downto 0) ;
               es: in std_logic_vector(4 dcwnto 0) ;
             pcout: inout std_logic_vector (7 downto 0));
end component;

component stack port (
               clk: in std_logic;
             pcout: in std_logic_vector (7 downto 0);
             reset: in std_logic;
              pila: inout std_logic_vector (7 downto 0);
                es: in std_logic_vector (4 downto 0) ) ;
end component;

component Índice port (
               clk: in std_logic;
          pcontrol: in std_logic_vector (7 downto 0);
            reset:in std_logic;
               es: in std_logic_vector(4 downto 0);
               ix: inout std_logic_vector (7 downto 0)) ;
end component;

component sel_dir port (
             pcout:in std_logic_vector (7 downto 0);
               clk:in std_logic;

                ix:in std_logic_vector (7 downto 0);
         dirección:out std_logic_vector (7 downto 0);
                cs:in std_logic_vector (4 downto 0));
end component;


component dir_val port (
               vma:out std_logic;
```

Continúa

```
                        cs:in std_logic_vector (4 downto 0) ) ;
    end component;

    component control port (
            reset,clk: in std_logic;
                  irq: in std_logic_vector (1 downto 0) ;
                   rw: inout std_logic;
              datoin: in std_logic_vector (3 downto 0);



             pcontrol: inout std_logic_vector (7 downto 0);
                   re: in std_logic_vector (3 downto 0) ;
                peout: in std_logic_vector (7 downto 0);

                   cs: inout std_logic_vector (4 dcwnto 0) ) ;
    end component;
end cmicro;
```

Listado 8.12 Creación de los componentes del diseño.

De acuerdo con el código anterior, cada componente se declara de manera similar a su entidad de diseño y cada uno se almacena en el paquete llamado *cmicro.*

Creación del archivo de alto nivel

Según se mencionó en el capítulo 2, en VHDL es posible diseñar en forma estructural; es decir, uniendo componente por componente. Esta metodología es la base del diseño jerárquico, además cada bloque o componente del diseño se interconecta por medio de señales o buses internos, los cuales se declaran y asocian mediante cláusulas propias del lenguaje. Con ellos se completa el diseño de bloques funcionales que interconectados forman sistemas complejos.

La declaración de la entidad consiste en todas las terminales de entrada y salida del microprocesador, las cuales se nombran tal como se encuentran referidas en su módulo. Las señales que interconectan cada componente se declaran en la arquitectura.

El código en VHDL del archivo de alto nivel (high level) que permite trabajar en conjunto a todos los bloques del microprocesador aparece en el listado 8.13.

```
u4: sel_dato port map (cs=>cs, datoin=>datoin, datout=>datout, operacion=>operacion, clk=>clk);

u5: tri_esr port map (cs=>cs, operacion=>operación, datout=>datout);

u6: pe port map (clk=>clk, pcontrol=>pcontrol, cs=>cs, pcout=>pcout, pila=>pila);

u7: Índice port map (cs=>cs, clk=>clk, pcontrol=>pcontrol, ix=>ix);

u8: stack port map (cs=>cs, clk=>clk, pcout=>pcout, pila=>pila);

u9: sel_dir port map (clk=>clk, pcout=>pcout, ix=>ix, direccion=>direccion, cs=>cs);
ul0: dir_val port map (vma=>vma, cs=>cs);

ull: control port map (clk=>clk, reset=>reset, irq=>irq, rw=>rw, datoin=>datoin, pcontrol=>pcontrol, rc=>rc, pcout=>pcout, cs=>cs);
end;
```

Listado 8.13 Programa de alto nivel.

Lista de instrucciones

El microprocesador contiene un conjunto de 16 instrucciones, dichas operaciones pueden ocupar 1, 2 o 3 *nibbles* según el tipo de operación que se va a ejecutar. En la tabla 8.2 se puede observar el conjunto de instrucciones. Al final de dicha tabla se describe cada uno de los símbolos que se utiliza en ésta.

Instrucción	Mnemónico	~	#	RCC NZVC	Código de operación	Comentario
A + B—*A	ABA	6	1	• • • •	0000(0)	Suma los acumuladores y guarda el resultado en A.
A - B — A	SBA	5	1	• • • •	0001(1)	Resta los acumuladores y guarda el resultado en A
A and B — A	ANAB	5	1	• • • •	0010(2)	AND lógica entre acumuladores y guarda el resultado en A.
A or B—•A	ORAB	5	1	• • • •	0011(3)	OR lógica entre acumuladores y guarda el resultado en A.
not A — *A	NEO	5	1	• • • •	0100(4)	Invierte A y guarda el resultado en A.
A xor B—-"A	EOAB	5	1	• • • •	0101(5)	XOR lógica entre acumuladores y guarda el resultado en A.

Tabla 8.2 Lista de instrucciones del microprocesador.

Instrucción	Mnemònico		#	RCC NZVC	Código de operación	Comentario
A—•M	STTA	4	1	lilí	1001(9)	Almacena el contenido de A en la dirección del índice.
X + 1—X	INX	3	1	111!	1010(A)	Incrementa en una unidad el valor del registro índice.
C = "1"	BCS	3, 6	2	1111	101I(B)	Salta siC = '1'.
Z = "1"	BEQ	3,6	2	111í	1100(C)	Salta siZ = T.
N = "1"	BMI	3,6	2	1111	1101 (D)	Salta siN = T.
V = "1"	BVS	3,6	2	1111	1110(E)	Salta si V = '1'.
PILA — Pcout	RTÍ	3	1	1111	1111 (F)	Regresa de una interrupción.
M—»X	LDX M	7	3	1111	0110(6)	Carga el índice con la dirección de memoria inmediata.
M—» A	LDDA	5	2	1111	0111(7)	Carga el acumulador A con el valor inmediato.
M—•B	LDDB	5	2	‡ł¡i!f	1000(8)	Carga el acumulador B con el valor inmediato.

Tabla 8.2 Lista de instrucciones del microprocesador. *(Continuación.)*

Descripción de los símbolos utilizados en la tabla 8.2:

y —• Transferencia a.
> • Se pone en "1" si es cierto y en caso contrario en "0"
> A Acumulador A.
> B Acumulador B.
> X registro de índice.
> C Indica si hay acarreo.
> Z Señala que el dato es igual a cero.
> V Indica si hay sobreflujo.
> N Señala que el dato es negativo.
> RCC Registro de códigos de condición.
> # Número de mordiscos*(nibbles)* requeridos por la instrucción.
> ~ Cantidad de ciclos de reloj empleados por la instrucción.
> M Dato de una dirección de memoria o una dirección de memoria.
> J No afectado.

De acuerdo con la tabla 8.2, el usuario sólo puede manipular los acumuladores y el registro cié índice, además de utilizar el registro de códigos de condición para conocer el resultado; no puede modificar el resto de los registros del microprocesador.

Ejercicios

Programación de sistemas embebidos

8.1 A continuación se describe el sistema embebido 14500 de Motorola[1], que es un procesador programable de un bit ideado para construir sistemas de control relativamente sencillos, en los que preferentemente se requieren decisiones lógicas en vez de cálculos numéricos. Una aplicación típica del 14500 en el control de procesos industriales es el conectar o desconectar algunos dispositivos como respuesta a cambios en el medio. Por ejemplo, si el conmutador SI de la alimentación principal está conectado, un conmutador S2 limitador de movimiento está desconectado y también está desconectado un conmutador termostático S3, entonces debe conectarse el motor M. Operaciones de control lógico de este estilo hace unos años se realizaban frecuentemente con circuitos de relés electromecánicos y, más recientemente, mediante circuitos lógicos cableados o no programables, con dispositivos SSI/MSI. El 14500 y sus circuitos de apoyo están pensados para proporcionar una alternativa de bajo costo a los circuitos lógicos cableados en aplicaciones industriales del estilo anterior. Por esta razón el 14500 se conoce como unidad de control industrial (ICU) en las referencias de los fabricantes.

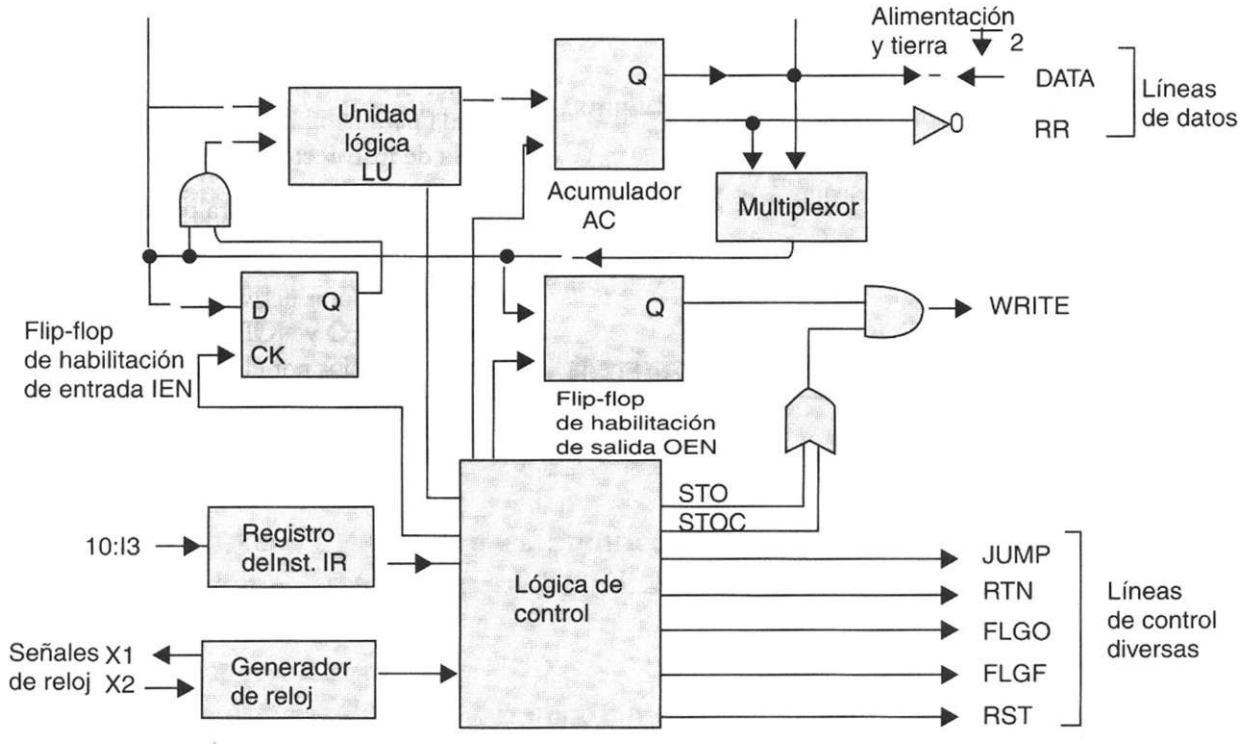
Descripción del circuito

El 14500 es básicamente la unidad E de un microprocesador sencillo de 1 bit. Dado su tamaño de palabra tan corto y el conjunto relativamente pequeño de instrucciones (sólo se utilizan 16 instrucciones), el 14500 se puede encapsular con solo 16 patas. En la figura 8.1. se muestra su estructura interna. El principal elemento para el procesamiento de datos en ICU es la unidad lógica LU, que corresponde a la unidad aritmético-lógica de los procesadores convencionales. Como reflejo de las aplicaciones para las que está ideado, el 14500 tiene instrucciones lógicas, pero no instrucciones aritméticas explícitas. La LU recibe sus operandos de entrada de un acumulador AC de 1 bit (realmente un flip-flop D) y de una línea de datos bidireccional denominada DATA; esta línea actúa como principal bus de datos de la ICU. Los resultados obtenidos por la LU se almacenan en AC, de donde se pueden transferir a cualquier destino externo a través de la línea DATA. Los contenidos de AC están disponibles continuamente en la pata de salida denominada RR. (RR quiere decir registro de resultados, que es el nombre que Motorola utiliza para el acumulador). Además de LU y AC, el 14500 tie-

[1] John R Hayes. *Diseño de sistemas digitales y microprocesadores.* Prentice Hall, Págs. 405-409.

ne un reloj, un registro de instrucción IR de 4 bits y circuitos lógicos que generan una variedad de señales de control y de temporización utilizadas para comunicar con los dispositivos externos incluyendo la unidad I, la memoria principal y dispositivos de E/S.

EL 14500 tiene 16 terminales o patas de E/S, cuyas funciones se definen en la tabla 8.1. Las operaciones a realizar por la ICU se especifican mediante una instrucción de 4 bits que se aplica a las patas de entrada 10 a 13. como reloj del sistema se puede utilizar una señal de reloj que se tiene en la pata de salida XI; también la ICU puede estar controlada mediante una señal de reloj externa conectada a la pata de entrada X2. Para el 14500 se utiliza tecnología de circuitos CMOS, que requiere una pata para alimentación (+5V) y otra para tierra. Una señal de reset, RST, sirve para poner todos los registros a sus valores iniciales. Se utiliza la señal de salida WRITE para habilitar la memoria externa durante la ejecución de las instrucciones de almacenar. Típicamente las cuatro señales de control JMF| RTN, FLGO y FLGF se transmiten a la unidad externa I durante la ejecución de las instrucciones de control del programa. La temporización de las instrucciones es muy simple. Una vez por cada ciclo de reloj las señales en 10:13 se transfieren a IR; a continuación se decodifican y ejecutan.



Considerando a este microprocesador como un sistema embebido,

a) Programe a través de una serie sucesiva de procesos para cada entidad individual
b) Programe mediante una integración global
c) Programe mediante top level

Tipo	Nombre	Función
Datos	DATA	Línea bidireccional de datos para entrada y salida
	RR	Línea de salida desde el acumulador (registro de resultado)
Instrucción	10:13	Líneas de entrada de instrucción (4 bits)
Alimentación	V_DD	Alimentación (+5V)
	v_{ss}	Tierra (0 V)
Temporización	XI	Señales de control del sistema generada por el 14500
	X2	A X2 se puede conectar una señal de reloj externa También puede conectarse una resistencia entre XI y X2 para fijar la frecuencia del reloj interno.
Controles	WRITE	Señal activada por las instrucciones de almacenamiento. Utilizada para habilitar la escritura de memoria externa
	JMP	Señal activada por la instrucción de salto. Utilizada para cargar un contador de programa externo como una dirección de salto
	RTN	Señal activada por la instrucción de retorno desde la subrutina. Utilizada para habilitar una operación de extracción en una pila de instrucciones externa.
	RST	Señal de reset utilizada para poner a cero todos los registros
	FLOGO.FLGH	Señales indicadoras activadas por las instrucciones de no operación ÑOPO y NOPF, respectivamente. Varios posibles usos definidos por el usuario.

Bibliografía

Ganssle Jack. *The Art of Designing Embedded Systems.* Ed. Newnes. USA, 1999.

Valvano W. Jonathan. *Embedded Microcomputer Systems Real Time Interfacing.* Ed. Brooks/Cole. Ca. USA, 2000.

Powel Douglass. *Real Time UML Developing Efficient Objects for Embedded Systems.* Ed. Addison-Wesley, 1998.

Kenneth L. Short. *Embedded* Microprocessor *Systems Design.* Ed. Prentice Hall, 1998.

G. Maxinez David, Alcalá J. Jessica. *Diseño de secuenciadores integrados utilizando campos de lógica programable FPGA.* Congreso Internacional Académico Electro'98. México, 1998.

Skahill Kevin. *VHDL For Programmable Logic.* Addison Wesley, 1996.

Mazor Stanley, Langstraat Patricia. A *Guide to VHDL.* Kluwer Academic, 1992.

T.L. Floyd. *Fundamentos de Sistemas Digitales.* Prentice Hall, 1998.

Ll. Teres; Y. Torroja; S. Olcoz; E. Villar. *VHDL, lenguaje estándar de diseño electrónico.* McGraw-Hill, 1998.

David G. Maxinez, Jessica Alcalá. *Diseño de Sistemas Embebidos a través del Lenguaje de Descripción en Hardware VHDL.* XIX Congreso Internacional Académico de Ingeniería Electrónica. México, 1997.

C. Kloos, E. Cerny. *Hardware Description Language and their Applications. Specification, Modeling, Verification and Synthesis of Microelectronic Systems.* Chapman & Hall, 1997.

IEEE. *The IEEE standard VHDL Language Reference Manual.* IEEE-Std-1076-1987. 1988.

Zainalabedin Navabi. *Analysis and Modeling of Digital Systems.* McGraw-Hill, 1988.

R J. Ashenden. *The Designer's guide to VHDL.* Morgan Kauffman Publishers, Inc., 1995.

R. Lipsett, C. Schaefer. *VHDL Hardware Description and Design.* Kluwer Academic Publishers, 1989.

J.R. Armstrong y F. Gail Gray. *Structured Design with VHDL.* Prentice Hall, 1997.

J. A. Bhasker. A *VHDL Primer.* Prentice Hall, 1992.

H. Randolph. *Applications of VHDL to Circuit Design.* Kluwer Academic Publisher.

www.imagecraft.com

www.cce.utexas.edu/~valvano/book.html

http://www2.unam.edu.ar/subprograma/metod_anexl.htm

Capítulo 9

Redes neuronales artificiales y VHDL

Introducción

Las redes *neuronales artificiales* son modelos de comportamiento inteligente que pueden ser construidos como sistemas artificiales inspirados en el sistema nervioso de seres vivos. Las aplicaciones de sistemas basados en redes neuronales han permitido diseñar redes con propósitos específicos como el reconocimiento de patrones. Este esquema es novedoso en el área de la computación y es muy interesante dado que una computadora digital, aun la más sencilla, supera la velocidad y precisión del cerebro en la relación de operaciones numéricas; aunque las operaciones como reconocimiento de patrones, memoria asociativa y en general todas las relaciones con el comportamiento inteligente parecen imposibles de alcanzar mediante las concepciones computacionales tradicionales de la computación incluyendo el campo de la inteligencia artificial.

Estas similitudes de los modelos de las redes neuronales y el funcionamiento del cerebro parecen sugerir que la posibilidad de conocer más sobre el comportamiento de sistemas inteligentes —o más ambiciosamente, la reproducción de ciertas propiedades de la inteligencia en "cerebros artificiales" mediante redes neuronales artificiales. El desarrollo de sistemas artificiales inteligentes se puede obtener no sólo con algoritmos más perfeccionados, sino con cambios más o menos radicales en la concepción básica de las estructuras computacionales.

Existe una gran cantidad de problemas de ingeniería resueltos con diseños de redes neuronales artificiales capaces de emular en mayor o menor medida algunas funciones realizadas por el sistema nervioso de los seres vivos, como la discriminación de patrones. No obstante el éxito obtenido por las redes neuronales artificiales, no está comprobado que esta arquitectura

computacional cumpla con el test de Turin y que, por consiguiente, sea la solución al problema de la generación de verdadera inteligencia artificial. Desde el punto de vista filosófico, las arquitecturas distribuidas y paralelas, como las redes neuronales, no son más que muchos procesadores tradicionales; desde este punto de vista, no poseen alguna propiedad nueva que evite las objeciones del test de Turin.

Sin embargo, una gran cantidad de grupos proporciona evidencias de que estas redes poseen nuevas propiedades, algunas muy similares a las que tiene el funcionamiento básico del cerebro.

Para demostrar que las redes neuronales son capaces de aprender y, en fin, tener un funcionamiento que se aproxime al comportamiento de la mente y ser la base de la inteligencia artificial, se aplica la prueba más importante basada en el conocimiento de que no basta ser como un cerebro, en cuanto a la arquitectura se refiere, sino que hay que comportarse como tal para emular sus propiedades básicas. Esto significa que la red debe funcionar también en forma análoga a como lo hace el sistema nervioso. En esta línea de pensamiento se han dado pasos trascendentales que se han materializado en microchips, algunas aplicaciones inspiradas en las estructuras de sistemas biológicos que realizan funciones de visión o auditivas como la retina artificial y la sustitución de la función parcial de la cóclea con circuitos integrados.

Las redes neuronales artificiales inspiradas en el funcionamiento del sistema nervioso de seres vivos han permitido a las computadoras modernas emprender bien definidas mediante algoritmos de redes neuronales, como auxiliares auditivos y transductores de visión de amplio espectro. Las computadoras digitales de arquitectura tradicional abordan ese tipo de tarea con mayores dificultades que las de arquitecturas de redes celulares neuronales (CNN) que pueden ser diseñadas con arreglos de bloques en VHDL. Esta diferencia ha motivado a las comunidades científicas a estudiar sistemas neuronales biológicos en un esfuerzo por diseñar sistemas computacionales con capacidades semejantes a las del cerebro. Así, la tecnología de circuitos integrados modernos está ofreciendo su potencialidad para construir redes paralelas masivas con base en elementos simples de procesamiento. De esta manera, la disciplina formal del estudio de redes neuronales, la **neurocomputación,** sienta las bases necesarias para programar y coordinar el comportamiento de dichos elementos de procesamiento.

Los modelos de redes neuronales están ofreciendo nuevos enfoques para resolver problemas, mismos que pueden simular en máquinas de propósito específico, aceleradores de hardware con arquitectura neuronal incluso en computadoras convencionales. A fin de alcanzar velocidades de procesamiento máximo pueden utilizarse arreglos con base en tecnología de procesamiento óptico o en VLSI de silicio. El aspecto fundamental reside en utilizar el modelo neuronal de procesamiento paralelo en tiempo real con una enorme cantidad de procesadores sencillos.

9.1 ¿Qué es una red neuronal artificial?

Las redes neuronales artificiales son sistemas basados en una representación simplificada de estructura y funcionamiento del sistema nervioso, ya sea simulado en software o construido en hardware. Se deben entrenar mediante ejemplos conocidos hasta que son capaces de asociar patrones de entrada con respuestas definidas sin necesidad de una programación explícita para un patrón en particular. Esta característica permite a las redes resolver problemas nuevos donde incluso se crean nuevas categorías de patrones para los cuales no es posible diseñar algoritmos de solución, con lo cual pueden enfrentar con éxito casos en que no es posible definir una relación entre las causas y los efectos.

Las redes neuronales son modelos de cómputo paralelo que se conocen en ingeniería con el nombre de redes neuronales artificiales. Ofrecen una alternativa de procedimientos a problemas cuya solución por técnicas tradicionales resulta de difícil aplicación en la mayoría de los casos.

9.1.1 Funcionamiento de la red

El principio de funcionamiento de las redes neuronales artificiales es, en términos generales, el de un convertidor vectorial. La información codificada en forma de vector de entrada controla la red y después de cierta cantidad de intentos de comparación entre patrones ésta produce un vector de salida, que es la mejor solución que encuentra en la conversión del vector de entrada. Así, las distintas respuestas de la red dependen de su arquitectura, de sus propiedades de conectividad y de las reglas algorítmicas explícitas en el paso de información de una capa de neuronas a las siguientes. El mejoramiento de las respuestas generadas por la red depende del entrenamiento de ésta mediante el cual cambia —en función de un algoritmo de aprendizaje o retroalimentación, también depende de la forma en que pasa la información entre las neuronas de la red y entre las capas de éstas.

De este modo la red realiza algunas funciones elementales similares a las que realiza el cerebro en tiempos razonables aunque no en tiempo real. Sin embargo, la implementación aún es similar a la de la inteligencia artificial clásica, pues todas estas redes se instalan en computadoras secuenciales con algoritmos que se ejecutan paso a paso y no de manera simultánea como ocurriría en una arquitectura en paralelo donde cada neurona de la red resuelve independientemente la función que tiene asignada.

Similitudes entre una red neuronal artificial y una neurona biológica

Para comprender mejor qué son las redes neuronales artificiales, es necesario hacer una breve referencia de la relación y similitudes que tienen éstas con las neuronas biológicas.

El funcionamiento de las redes neuronales se basa en el del sistema nervioso. En general, las criaturas vivientes tienen receptores que utilizan para ver, oler, oír y sentir.

Con base en el desarrollo de las redes neuronales, se han construido circuitos electrónicos que pueden aprender de manera similar a como lo hacen los seres vivos. El fundamento del aprendizaje está en el funcionamiento de la celda nerviosa básica de la neurona biológica. Estos pequeños bloques o estructuras de todos los sistemas vivientes, tanto del cerebro como del sistema nervioso de casi cualquier ser vivo, es posible modelarlos mediante un filtro transversal cuyos pesos varían o se adaptan cuando han alcanzado el aprendizaje deseado. A esta estructura se le conoce también como filtro transversal adaptable.

En la figura 9.1 tenemos el modelo de dos neuronas biológicas. La neurona del sistema nervioso está compuesta por un cuerpo celular y sus prolongaciones (dendritas y axón). Esto es, una neurona tiene muchas entradas y una salida. Las dendritas en la neurona equivalen a los receptores de información proveniente de otra neurona. Dicha información llega al cuerpo celular y de ahí se envía por medio del axón hacia otras neuronas. Este proceso de comunicación entre neuronas se llama sinapsis.

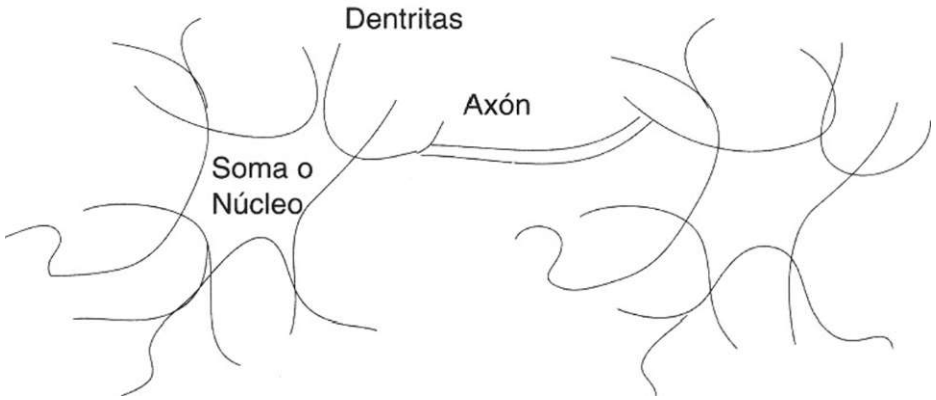


Figura 9.1 Modelo de una neurona biológica.

La entrada de una red neuronal biológica puede estar compuesta por un conjunto de sensores que desempeñan la función de receptores de datos del patrón, los cuales proporcionan los estímulos provenientes del exterior al interior de la red; dichos estímulos están representados por impulsos eléctricos que se encargan de transmitir la información hacia el sistema nervioso central, provocando efectos de respuesta humana que se puede expresar en una gran variedad de acciones. Al igual que en la neurona, en los circuitos electrónicos existe un estado bajo o alto (0 o 1), las entradas a los circuitos se

acumulan y luego se procesan por medio de una función de activación que determina la respuesta de la neurona. Para la función de activación suele utilizarse una función matemática que varía en forma continua con un perfil sigmoidal. La salida de la neurona o rango dinámico cambia entre los límites de 0 y 1. Esto significa que la neurona transmitirá información cuando el nivel 1 es sobrepasado pero no para valores inferiores, pues es un sistema de todo o nada en su respuesta.

Tanto en la neurona biológica y en los circuitos electrónicos la comunicación se establece mediante un pulso eléctrico. En la célula nerviosa podemos medir el voltaje de la membrana, que es cercano a 0.1 volt. En los alambres conductores también se puede medir el voltaje cuando circula la corriente eléctrica. La transmisión del impulso nervioso se propaga en la célula a una velocidad que oscila entre 27 y 132 metros por segundo. Así también las corrientes en los alambres conductores se pueden propagar (por efectos de la acción de los campos) a una velocidad cercana a la de la luz (300 000 km/seg).

9.1.2 Aplicaciones de redes neuronales artificiales

Las aplicaciones de las redes neuronales y de la computación son muy abundantes. De entre ellos se destacan en el campo de procesamiento de señales o reconocimiento de patrones, la extracción de características, la inspección industrial, el pronóstico de negocios, la clasificación de crédito, la selección de seguridad, el diagnóstico médico, el procesamiento de voz, la comprensión del lenguaje natural, el control de robots y la adaptación de procesos de control.

Elementos de una red neuronal

Las redes neuronales artificiales constan de elementos de procesamiento y conexión de pesos. La cantidad de éstos depende de la implementación y diseño de la red. Otros elementos que intervienen dependiendo de la aplicación son las funciones preestablecidas de los vectores de entrada y los vectores de salida.

Elementos de entrada y salida

Acerca de los componentes de una red neuronal artificial es conveniente dar a conocer la terminología que se utilizará en la descripción de cada elemento.

En la figura 9.2 se observa la estructura de una red neuronal artificial. Consta de núcleo o nodo, entradas y salidas. La entrada es una señal de voltaje aplicada a un nodo, la cual puede ser una salida. Esta última es la respuesta

de la neurona artificial. El núcleo o nodo suma las señales de entrada, las procesa y da una respuesta.

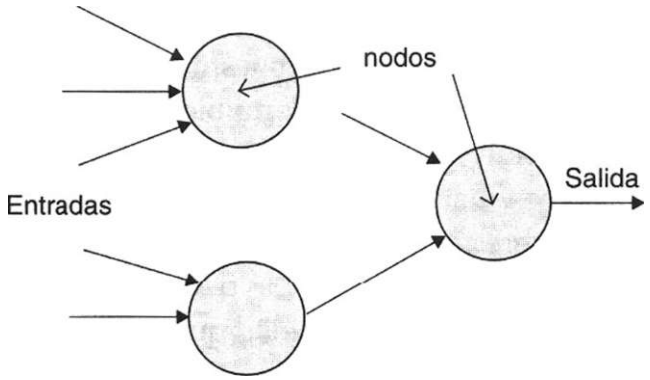


Figura 9.2 Red neuronal artificial.

En la figura 9.3 tenemos la estructura básica de una neurona artificial. Las entradas se denotan con las variables x_i y En algunas ocasiones se puede utilizar la notación vectorial x, que representa un vector con valores $(.x_i\ x_2)$.

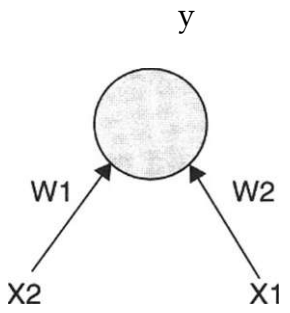


Figura 9.3 Estructura básica de una neurona artificial.

La salida se denota con y. Si bien en este caso sólo se tiene una salida, en una red neuronal se pueden tener diferentes salidas. Si esto ocurre, se denota con un vector $y = (y_i + y2 + ...)$.

Los pesos conectados entre los niveles de la red neuronal sirven para modificar los valores que circulan en las conexiones y son almacenados en un arreglo matricial denominado matriz de ponderaciones o pesos, por lo general representada por la letra w. En la figura 9.3, la matriz de pesos es un vector (un caso particular de una matriz) w con valores (wj w_2).

9.2 Aprendizaje en las neuronas artificiales

9.2.1 Neurona hebbiana

En 1949, Donald Hebb mencionó la primera regla de aprendizaje para una red neuronal artificial, la cual establece que cuando una neurona estimula a otra, la conexión entre ellas se refuerza.

$$I = b + \sum_{i=1}^{n} w_i x_i = b + \mathbf{w} \cdot \mathbf{x}$$

$$y = \begin{cases} +1; & \text{si } I \geq 0 \\ -1; & \text{si } I < 0 \end{cases}$$

$$\mathbf{w}_{\text{nueva}} = \mathbf{w}_{\text{anterior}} + y\,\mathbf{x}$$

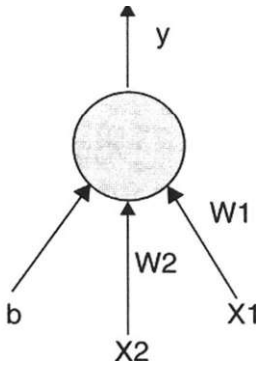


Figura 9.4 Ecuaciones características y modelo de una neurona Hebbiana.

En la figura 9.4 se tienen las ecuaciones generales y el modelo de una neurona hebbiana. El funcionamiento de ésta es muy sencillo. La neurona artificial calcula la entrada ponderada I.

$$I = b + \sum_{j=1}^{n} w_j x_j = b + w \cdot x$$

Donde w es la matriz de pesos, x es el vector de entrada y b es una constante inicializada en 1.

Si el valor de la entrada ponderada es mayor o igual a cero, la salida de la neurona es igual a +1; si es menor a cero, la salida es —1.

$$y = \begin{cases} +1; & \text{si } I \geq 0 \\ -1; & \text{si } I < 0 \end{cases}$$

Toda red neuronal se debe entrenar para que responda como se desee. En otras palabras, es necesario calcular el vector de ponderaciones (pesos) w que le permita comportarse de acuerdo con la información de entrada.

El proceso de entrenamiento para una neurona hebbiana es muy simple. Basta cambiar el vector de ponderaciones w, por cada patrón de entrenamiento, de acuerdo con la siguiente regla hebbiana:

$$\mathbf{w}_{\text{nueva}} = \mathbf{w}_{\text{anterior}} + y\,\mathbf{x}$$

Entrenamiento de una neurona hebbiana

Algoritmo de entrenamiento para la neurona hebbiana

```
Inicializar el vector de ponderaciones w en cero.
Para cada patrón de entrenamiento:
{
 Igualar el vector de entrada x con el patrón de entrenamiento actual.
 Igualar la salida y con la salida deseada;
 Ajustar el vector de ponderaciones w como
        Nuevo w = anterior w + x y;
 Ajustar
        Nuevo b = anterior b + y;
}
```

Ejemplo 9.1 Entrenar una neurona hebbiana para reconocer la compuerta AND.

Entrada		Salida
(	x_2)	
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

Solución

Para el primer vector de entrenamiento (1, 1), tenemos:

$$\mathbf{x} = (1, 1),\ x_1 = 1,\ x_2 = 1,\ y = 1,\ \mathbf{w}_0 = (0, 0),\ b_0 = 1$$

calculando el siguiente vector de ponderaciones de acuerdo con la ecuación 9.2:

$$\mathbf{w}_1 = \mathbf{w}_0 + \mathbf{x}\, y = (1, 1)$$

$$b_1 = b_0 + y = 2$$

Para el segundo patrón de entrenamiento (1,-1):

se calcula el siguiente vector de ponderaciones

Para el tercer patrón de entrenamiento (-1, 1):

se calcula el siguiente vector de ponderaciones

```
1 use std.textio.all;

3 ENTITY hebb IS END;
```

Para el último patrón de entrenamiento (-1,-1):

```
5 ARCHITECTURE neurona OF hebb IS
6   FILE archivo_entrada: TEXT OPEN READJMODE IS "patron.in";
7   FILE archivo_salida: TEXT OPEN WRITE_MODE IS "pesos.out";
8 BEGIN
9   PROCESS
```

Continúa

se calcula el siguiente vector de ponderaciones

```
10      VARIABLE buf_in, buf_0ut: LINE;
11      VARIABLE xl,x2,y: integer range -1 to 1;
12      VARIABLE wl,w2: integer : = 0;
13      VARIABLE b: integer : = 1;
14   BEGIN
15     WHILE not endfile(archivo_entrada) LOOP
16          readline(archivo_entrada, buf_in);
17          read(buf_in, xl) ;
18          read(buf_in, x2);
19          read(buf_in, y) ;
20          wl : = wl + xl*y;
21          w2 : = w2 + x2*y;
22          b : = b + y;
23     END LOOP;
24     write (buf_out, wl) ;
25     write(buf_out,"   ");
26     write (buf_out, w2) ;
27     write(buf_out,"   ");
28     write(buf_out, b);
29     writeline(archivo_salida,buf_out);
30     WAIT;
31  END PROCESS;
32 END neurona;
```

Listado 9.1 Programa en VHDL para entrenar una neurona hebbiana.

En la línea 1 se carga la librería *textio.* Como es un programa de simulación, la entidad no contiene puertos de entrada o salida. En la línea 6 se declara el archivo del cual se leen los pesos y en la 7 el archivo donde se escribirán los pesos finales.

En las líneas 9 a 31 se declara el proceso que se ejecutará durante la simulación. Es importante hacer notar que el proceso no tiene lista sensitiva, por lo que es necesario declarar la instrucción *wait* para que termine.

En las líneas 10 a 13 se declaran las variables que se van a usar en el programa, como pesos (u^, <w-j), I y b.

En la línea 15 se tiene un ciclo para leer del archivo de entrada los patrones de entrenamiento x_2 y la salida esperada y; al mismo tiempo se calculan los nuevos valores de los pesos y de la *constante b.*

Finalmente de la línea 24 a 29 se mandan escribir los pesos al archivo de salida mediante las instrucciones *write y writeline.*

Ejemplo 9.2 Se desea que la neurona hebbiana reconozca y realice la operación lógica AND.

Solución

El archivo de entrada *patrón*.in contiene:

Uso de una neurona hebbiana

Algoritmo para una neurona hebbiana

```
Inicializar w
Inicializar b
Leer la entrada x
Calcular la entrada ponderada I
Si I > = 0, entonces
     salida es igual a +1
Si I<0, entonces
     salida es igual a -1;
```

El archivo de pesos pesos.out contendrá:

Este programa inicializa los pesos w y v con los valores encontrados en el algoritmo anterior. Se calcula la entrada ponderada con la ecuación 9.1 y se calcula el valor de la salida y.

Ejemplo 9.3 Demuestre la red hebbiana para el ejemplo 9.1.

Solución

Del ejemplo 9.1 se obtuvo w = (2, 2) y b = —1. Para el primer patrón (1,1) tenemos:

$x = (1, 1)$, $Xj = 1$, $x_2 = 1$
por lo tanto,
$I = xjWj + x_2w_2 + b = (1)(2) 4- (1)(2)\text{-}1 = 3$, como $I > 0$ entonces $y = 1$.

Para el primer patrón (1, 1) tenemos:
x = (1,-1), Xj = 1, x_2 = -1
por lo tanto,
I = XjWj + x_2w2 + b = (1)(2) + (—1)(2) —1 = -1, como I < 0 entonces y = —1.

Para el primer patrón (1,1) tenemos:
x = (-1, 1), xi = -1, x_2 = 1
por lo tanto,
I = $x_1 w_1$ + $x_2 w_2$ + b = (-1) (2) + (1)(2) -1 = -1, como I < 0 entonces y = —1.

Para el primer patrón (1,1) tenemos:
x = (-1, -1), Xj = -1, x_2 = -1
por lo tanto,
I = XJWJ + $x_2 w_2$ + b = (-1) (2) + (—1) (2)—1 = -5, como I < 0 entonces y = —1.

Con lo que las entradas se clasifican correctamente.

El listado 9.2 es el programa en VHDL para utilizar una neurona hebbiana con W[= 2, w_2 = 2 y beta = -1.

```
1 ENTITY hebb IS
2  PORT (xl, x2 : IN INTEGER RANGE -1 TO 1;
3         y : OUT INTEGER RANGE -1 TO 1) ;
4 END;

6 ARCHITECTURE neurona OF hebb IS
7 BEGIN
8   PROCESS (xl, x2)
9      CONSTANT wl, w2: INTEGER : = 2;
10     CONSTANT b: INTEGER : =  -1;
11     VARIABLE I: INTEGER;
12   BEGIN
13      I : =  b + xl*wl + x2*w2;
14      IF I> = 0 THEN
15         y< = 1;
16      ELSE
17         y< = -1;
18      END IF;
19   END PROCESS;
20 END neurona;
```

Listado 9.2 Programa para una neurona hebbiana entrenada para reconocer una compuerta lógica AND.

De las líneas 1 a la 4 se declara la entidad del programa; aquí se manifiestan como señales de entrada las variables x_i y que se dan como enteros entre -1 y 1. La señal de salida es y, que también es un entero entre -1 y 1.

Dentro de la arquitectura, encontramos el proceso que se activa con las señales de entrada xj y x¿. En la línea 9 hallamos la declaración e inicialización de los pesos. En la línea 10 se tiene la declaración e inicialización de la constante *b.* El cálculo de la entrada ponderada *I* se realiza en la línea 13. La decisión de activar la neurona se da a través de la condicional en las líneas 14 a 17.

9.2.2 Perceptrón

Warren S. McCulloch y Walter Pitts desarrollaron la red neuronal perceptrón en 1943 proponiendo las ecuaciones generales y el diagrama de la figura 9.5.

$$I = \sum_{i=1}^{n} w_i x_i = \mathbf{w} \cdot \mathbf{x}$$

$$y = \begin{cases} +1; & \text{si } I \geq \theta \\ -1; & \text{si } I < \theta \end{cases}$$

$$\beta = \begin{cases} +1; & \text{si la respuesta es correcta} \\ -1; & \text{si la respuesta es incorrecta} \end{cases}$$

$$\mathbf{w}_{\text{nueva}} = \mathbf{w}_{\text{anterior}} + \beta\, y\, \mathbf{x}$$

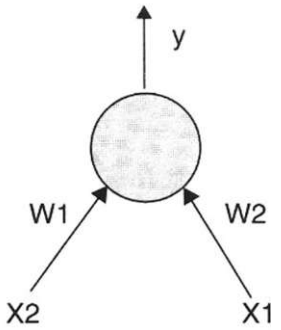


Figura 9.5 Ecuaciones generales y modelo del perceptrón.

El funcionamiento del perceptrón es muy simple. La neurona suma las señales del vector de entrada x, multiplicadas por el vector de ponderaciones (pesos) w, lo que da la entrada ponderada í.

$$I = \sum_{i=1}^{n} w_i x_i = \mathbf{w} \cdot \mathbf{x}$$

La ecuación compara la señal con un valor de umbral 0. Si es mayor, la salida es +1. De lo contrario, es -1.

$$y = \begin{cases} +1; & \text{si } I \geq \theta \\ -1; & \text{si } I < \theta \end{cases}$$

Para entrenar la red, se escogen valores de x como entrada, llamados patrones de entrenamiento. Por cada patrón de entrenamiento se calcula si la salida del perceptrón *y* es correcta o incorrecta, con lo que se obtiene fí.

$$\beta = \begin{cases} +1; & \text{si la respuesta es correcta} \\ -1; & \text{si la respuesta es incorrecta} \end{cases}$$

Por cada iteración el vector de ponderaciones w cambiará de acuerdo con la ley de Rosenblatt.

$$\mathbf{w}_{\text{nueva}} = \mathbf{w}_{\text{anterior}} + \beta \, y \, \mathbf{x}$$

Entrenamiento del perceptrón

Algoritmo de entrenamiento del perceptrón

```
para cada patrón de entrenamiento

{       calcular las entradas I;
        calcular la salida del perceptrón y;
        si es correcta, entonces
        {  si la respuesta es +1, entonces
               nuevo w = anterior w + el patrón de entrada actual;
           si la respuesta es -1, entonces
               nuevo w = anterior w - el patrón de entrada actual;
        }
        si es incorrecta, entonces
        {  si la respuesta es +1, entonces
               nuevo w = anterior w - el patrón de entrada actual;
           si la respuesta es -1, entonces
               nuevo w = anterior w + el patrón de entrada actual;
        }
```

Ejemplo 9.4 Entrene un perceptrón para reconocer la compuerta AND.

Entrada		Salida
(	x_2)	
1	1	1
1	- 1	- 1
- 1	1	- 1
- 1	- 1	- 1

Para el primer vector de entrenamiento (1, 1) tenemos:
$x = (1, 1), x_1 = 1, x_2 = 1, \mathbf{w}_0 = (0, 0), \theta = 1$
$I = w_1w_1 + w_2x_2 = (0)(1) + (0)(1) = 0$, como $I < \theta$ entonces $y = -1$.
La respuesta esperada es incorrecta; por lo tanto, $\beta = -1$, entonces
$\mathbf{w}_1 = \mathbf{w}_0 + \beta$ y $\mathbf{x} = (0, 0) + (-1)(-1)(1, 1) = (1, 1)$

Para el segundo vector de entrenamiento (1, -1) tenemos:
$\mathbf{x} = (1, -1), x_1 = 1, x_2 = -1, \mathbf{w}_1 = (1, 1), \theta = 1$
$I = w_1x_1 + w_2x_2 = (1)(1) + (1)(-1) = 0$, como $I < \theta$ entonces $y = -1$.
La respuesta es correcta, por lo que $\beta = 1$, entonces
$\mathbf{w}_2 = \mathbf{w}_1 + \beta$ y $\mathbf{x} = (1, 1) + (1)(-1)(1, -1) = (0, 2)$

Para el tercer vector de entrenamiento (1, -1) tenemos:
$\mathbf{x} = (-1, 1), x_1 = -1, x_2 = 1, \mathbf{w}_1 = (0, 2), \theta = 1$
$I = w_1x_1 + w_2x_2 = (0)(-1) + (2)(1) = 2$, como $I > \theta$ entonces $y = +1$.
La respuesta es incorrecta (se espera obtener –1); por lo tanto, $\beta = -1$, entonces
$\mathbf{w}_2 = \mathbf{w}_1 + \beta$ y $\mathbf{x} = (0, 2) + (-1)(1)(-1, 1) = (1, 1)$

Para el último vector de entrenamiento (1, -1) tenemos:
$\mathbf{x} = (-1, -1), x_1 = -1, x_2 = -1, \mathbf{w}_1 = (1, 1), \theta = 1$
$I = w_1x_1 + w_2x_2 = (1)(-1) + (1)(-1) = -2$, como $I < \theta$ entonces $y = -1$.
La respuesta esperada es correcta, así que $\beta = +1$, entonces
$\mathbf{w}_2 = \mathbf{w}_1 + \beta$ y $\mathbf{x} = (1, 1) + (1)(-1)(-1, -1) = (2, 2)$

El listado 9.3 es un programa en VHDL para entrenar un perceptrón con dos entradas (x_1, x_2) y una salida (y). El programa lee los patrones de entrenamiento de un archivo *patron.in* y escribe las ponderaciones (pesos) finales en un archivo *pesos.out.*

```
1 USE std.textio.all;
2
3 ENTITY perceptrón IS END;
4
5 ARCHITECTURE neurona OF perceptrón IS
6   FILE archivo_entrada: TEXT OPEN READ_MODE IS "patron.in";
7   FILE archivo_salida: TEXT OPEN WRITE_MODE IS "pesos.out";
8 BEGIN
9   PROCESS
10      VARIABLE buf_in, buf_0ut: LINE;
11      VARIABLE xl,x2,y: integer range -1 to 1;
12      VARIABLE wl,w2: integer : = 0;        —pesos iniciales
13      VARIABLE I: integer ;
14      VARIABLE respuesta: integer range -1 to 1;
15      CONSTANT theta : integer : = 1;
16   BEGIN
17      WHILE not endfile(archivo_entrada) LOOP
18       readline(archivo_entrada, buf_in);
19       read(buf_in, xl);
20       read(buf_in, x2);
21       read(buf_in, y);
22       I: =  wl*xl+w2*x2;
23       IF I> = theta THEN   —calcula la respuesta del perceptrón
24           respuesta: = 1;
25       ELSE
26           respuesta: = -1;
27       END IF;

29       IF y = respuesta THEN    --si la respuesta es correcta
30           IF respuesta = 1 THEN
31              wl: =  wl+xl;
32              w2: =  w2+x2;
33           ELSE
34              wl: =  wl-xl;
35              w2: =  w2-x2;
36           END IF;
37        ELSE                    —si la respuesta es incorrecta
38           IF respuesta = 1 THEN
39              wl: =  wl-xl;
40              w2: =  w2-x2 ;
41           ELSE
42              wl: = wl+xl;
43              w2: =  w2-x2;
44           END IF;
45        END IF;
```

Continúa

```
46      END loop;
47      write(buf_out, wl);    --se escribe los pesos finales
48      write(buf_out,"  ");   --en el archivo de salida
49      write(buf_out, w2);
50      writeline(archivo_salida,buf_out);
51      WAIT;
52   END process;
53 END neurona;
```

Listado 9.3 Programa en VHDL para entrenar el perceptrón.

Ahora se analizará el algoritmo en seudocógico que se presentó. En las líneas 22 a 27 se calcula la respuesta del perceptrón dados los diferentes patrones de entrada.

La modificación de los pesos, si la respuesta es correcta, se encuentra en el fragmento:

```
Si la respuesta es correcta, entonces
{        si la respuesta es +1, entonces
                nuevo w = anterior w + el patrón de entrada actual;
         si la respuesta es -1, entonces
                nuevo w = anterior w - el patrón de entrada actual;
}
```

el cual se encuentra en las líneas 29 a 36.

Si la respuesta es incorrecta se halla en el código:

```
si es incorrecta, entonces
  {      si la respuesta es +1, entonces
                nuevo w = anterior w - el patrón de entrada actual;
         si la respuesta es -1, entonces
                nuevo w = anterior w + el patrón de entrada actual;
  }
```

Y está codificada en las líneas 38 a 45.

Ejemplo 9.5 Se desea que el perceptrón reconozca y realice la operación lógica AND.

El archivo de entrada patrón.in contiene:

El archivo de pesos *pesos.out* contendrá:

```
2 2
```

Solución

Uso del perceptrón

El algoritmo para un perceptrón es muy sencillo. A continuación se muestra en seudocódigo.

Algoritmo para un perceptrón

```
inicializar w;
inicializar theta;
leer la entrada x;
calcular la entrada ponderada I;
si I> = 0, entonces
   salida es igual a +1;
si I<0, entonces
   salida es igual a -1;
```

Ejemplo 9.6 Compruebe que el entrenamiento del ejemplo 9.4 sea correcto.

Solución

De los ejemplos 9.4 y 9.5 se obtuvo $\mathbf{w} = (2, 2)$ y $\theta = 1$
Para el primer patrón (1, 1) tenemos:
$\mathbf{x} = (1, 1), x_1 = 1, x_2 = 1$
por lo tanto,
$I = x_1w_1 + x_2w_2 = (1)(2) + (1)(2) = 4$, como $I > \theta$, entonces $y = 1$.

Para el segundo patrón (1, -1) tenemos:
$\mathbf{x} = (1, -1), x_1 = 1, x_2 = -1$
por lo tanto,
$I = x_1w_1 + x_2w_2 = (1)(2) + (-1)(2) = 0$, como $I < \theta$, entonces $y = -1$.

Para el tercer patrón (-1, 1) tenemos:
$\mathbf{x} = (-1, 1), x_1 = 1, x_2 = -1$
por lo tanto,
$I = x_1w_1 + x_2w_2 = (-1)(2) + (1)(2) = 0$, como $I < \theta$, entonces $y = -1$.

Para el cuarto patrón (-1, -1) tenemos:
$\mathbf{x} = (-1, -1), x_1 = 1, x_2 = -1$
por lo tanto,
$I = x_1w_1 + x_2w_2 = (-1)(2) + (-1)(2) = -4$, como $I < \theta$ entonces $y = -1$.

Con lo que las diferentes entradas se clasifican correctamente.

El programa que clasifica los diferentes patrones de una compuerta AND mediante wj = 2, w_2 = 2 y theta = 1 se encuentra en el listado 9.4.

```
1 ENTITY perceptrón IS
2   PORT ( xl,x2: IN INTEGER RANGE -1 TO 1;
3          y: OUT INTEGER RANGE -1 TO 1);
4 END perceptrón;
5
6 ARCHITECTURE neurona OF perceptrón IS
7 BEGIN
8    PROCESS (xl,x2)
9       CONSTANT wl: INTEGER : = 2;
10      CONSTANT w2: INTEGER : = 2;
11      CONSTANT theta : INTEGER : = 1;
12      VARIABLE I: INTEGER;
13   BEGIN
14      I: = xl*wl+x2*w2 ;
15      IF I> = theta THEN
16         y< = 1;
17      ELSE
18         y< = -1;
19      END IF;
20   END PROCESS;
21 END neurona;
```

Listado 9.4 Programa en VHDL para el perceptrón entrenado para reconocer una compuerta lógica AND.

9.3 Aprendizaje por error mínimo (la Adaline)

Es un sistema que utiliza la técnica del mínimo error cuadrático, desarrollado en 1960 por Bernard Widrow y Ted Hoff, de la universidad de Stanford. Ofrece la primera neurona con un entrenamiento supervisado; es decir, toma en cuenta el posible error cuadrático mínimo obtenido al calcular la salida de la neurona. El nombre proviene de *adaptive linear element* (elemento lineal adaptativo) y en la figura 9.6 se muestra su modelo y ecuaciones.

$$I = \sum_{i=1}^{n} w_i x_i$$

$$y = \begin{cases} +1; & \text{si } I > 0 \\ -1; & \text{si } I \le 0 \end{cases}$$

$$\mathbf{w} = \mathbf{w}_{\text{anterior}} + \frac{\beta E \mathbf{x}}{|\mathbf{x}|^2} = \mathbf{w}_{\text{anterior}} + \Delta \mathbf{w}$$

E = <*Salida deseada*> - <*Salida actual*>;
es el error

β; es la constante de aprendisaje
(0 < β < 1)

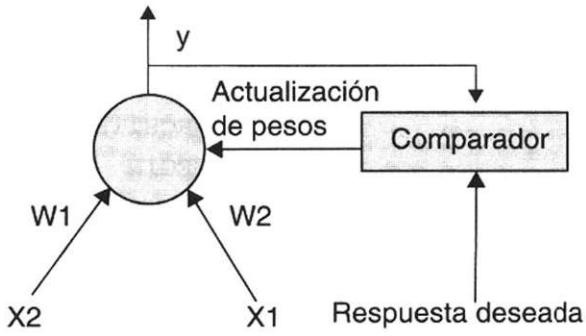


Figura 9.6 Modelo y ecuaciones características de la adaline.

La adeline, al igual que el perceptrón, tiene una salida bipolar. Esta genera en su salida + 1, si la entrada ponderada I es mayor que cero, y -1 si la entrada ponderada es menor o igual a cero. La salida se compara con la salida deseada calculando el error de la adaline *E.*

$$E = \langle \textit{salida deseada} \rangle - \langle \textit{salida actual} \rangle$$

Es evidente que el error sólo puede tomar los valores +2, -2 o cero. Una vez calculado el error, se puede utilizar para ajustar las ponderaciones w, utilizando la regla delta.

$$\mathbf{w} = \mathbf{w}_{\text{anterior}} + \frac{\beta E \mathbf{x}}{|\mathbf{x}|^2} = \mathbf{w}_{\text{anterior}} + \Delta \mathbf{w}$$

Donde (5 es la constante de aprendizaje y puede tomar un valor entre cero y 1. E es el error ya calculado, x es el vector de entrada (Xj, x_2) y w es el vector de ponderaciones (wj, w_2).

Entrenamiento de una adaline

El proceso de entrenamiento consiste en calcular el error para cada patrón de entrada. Si el error es diferente a cero, se modifica el vector de ponderaciones, pero después se debe regresar a los patrones anteriores para revisar que se siga obteniendo la salida deseada.

Algoritmo de entrenamiento para la adaline

```
Para cada patrón de entrenamiento
{  calcular la entrada I;
   calcular la salida de la adaline;
   calcular el error E;
   si E es diferente de 0, entonces
  {      para todos los patrones anteriores al patrón actual hacer
         {      calcular la entrada I;
                calcular la salida y;
                calcular el error E;
                mientras E es diferente de 0 hacer
                {      calcular L = Ixl^
                       para cada elemento de vector w hacer
                       {      cambiar dwi = ¡3 E xi / L;
                              vJi = w¿ + dw;- ;
                       }
                       calcular la entrada I con el nuevo w;
                       si I > 0 entonces
                              salida de la adaline es +1;
                       si I < = 0 entonces
                              salida de la adaline es -1;
                       calcular el error E;
                }
         }
  }
}
```

Uso de una adaline

Algoritmo para una adaline

```
inicializar el vector de pesos w;
leer la entrada x;
calcular la entrada ponderada I;
si I > = 0, entonces
        sal ida es igual a +1;
si I < 0, entonces
        salida es igual a -1;
```

9.4 Redes asociativas

En las redes asociativas los patrones se almacenan asociándolos con otros patrones. Existen varias clasificaciones para las redes asociativas; por ejemplo, pueden ser autoasociativas o heteroasociativas. Una red autoasociativa almacena los patrones asociándolos con ellos mismos. Una red heteroasociativa se utiliza para asociarlos con otros patrones.

Una red asociativa tiene la estructura de la figura 9.7.

$$I_j = \sum_i^n x_i w_{ij}$$

$$y_j = \begin{cases} 1; & \text{si } I > 0 \\ 0; & \text{si } I = 0 \\ -1; & \text{si } I < 0 \end{cases}$$

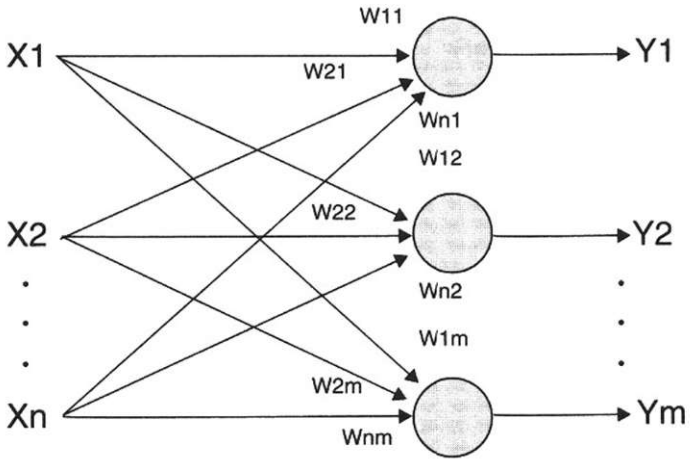


Figura 9.7 Ecuaciones características y modelo de la red asociativa.

El funcionamiento de la red es el siguiente: se calculan las entradas ponderadas a cada una de las neuronas.

$$I_j = \sum_i^n x_i w_{ij}$$

La función de transferencia para este tipo de redes está dada por la siguiente ecuación si las salidas son de tipo bipolar:

$$y_j = \begin{cases} 1; & \text{si } I > 0 \\ 0; & \text{si } I = 0 \\ -1; & \text{si } I < 0 \end{cases}$$

Si las salidas son de tipo binario, se puede utilizar una función como la siguiente:

$$y = \begin{cases} 1; & \text{si } I > 0 \\ 0; & \text{si } I \leq 0 \end{cases}$$

Ley de Hebb para asociación de patrones

La ley de Hebb es la regla más simple y el método más usado para determinar los pesos de una red asociativa. Se puede usar con patrones representados por vectores binarios (0, 1) o bipoles (−1, 1). En este caso se denotan los pares de vectores de entrenamiento como e: s. Donde e es el vector de entrada y s es el vector de salida esperado para el vector de entrada.

Entrenamiento de una red heteroasociativa mediante la ley de Hebb

El algoritmo para entrenar una red heteroasociativa es muy sencillo: para cada uno de los patrones de entrenamiento se utiliza la ley de Hebb ya mencionada. Cada peso se almacena en una matriz.

Algoritmo de entrenamiento para una red heteroasociativa mediante la ley de Hebb

```
Inicial i zar los pesos w¿j = 0;
Para cada patrón de entrenamiento e : s
{ Hacer las entradas de activación igual al patrón de entrada de entrenamiento,
        x¿ = e¿
  Hacer las salidas de activación igual al patrón de salida de entrenamiento,
        Y/ = sj
  Ajustar los pesos:
        wij (nueva) = wij (anterior) + xiyi
}
```

Ejemplo 9.7 Entrene a una red heteroasociativa mediante la ley de Hebb. Suponga que el vector de entrada e = (ej, e_2, e_3, e_4) y el vector de salida s = (s_{I5} s_2) son:

e_1	e_2	e_3	e_4	s_1	s_2
1	0	0	0	1	0
1	0	1	0	1	0
0	1	0	1	0	1
0	0	0	1	0	1

Solución

La topología de la red heteroasociativa se muestra en la figura 9.8.

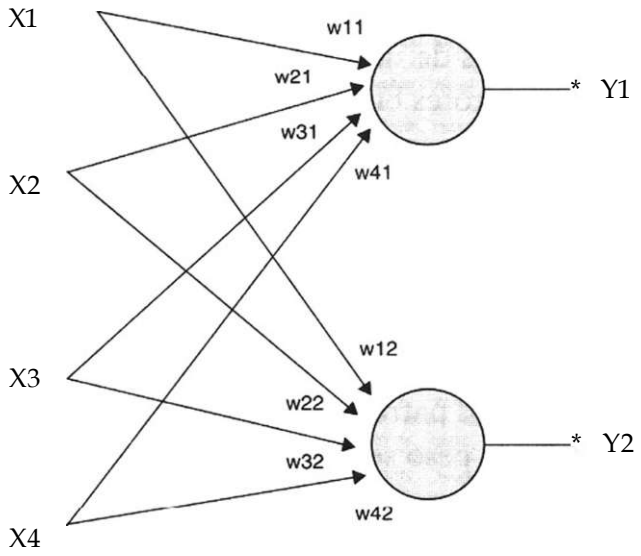


Figura 9.8 Red heteroasociativa para el ejemplo 9.7.

El resultado de aplicar el algoritmo anterior es el siguiente:

Inicializar los pesos en cero.

Para el primer par de vectores de entrenamiento e : s, (1 0 0 0):(1 0)
X] = 1, x_2 = x_3 = X4 = 0
Yl — 1, Y2 = 0
nueva w^ = anterior w^ 4- xj yj = 0 + (1)(1) = 1
los demás pesos se mantienen en cero.

Para el segundo par de vectores de entrenamiento e : s, (1 0 1 0):(1 0)
X! = 1, x_2 = o, X3 = 1, X4 = o
Yl = 1. Y2 = 0
nueva W[[= anterior wjj + x^ y^ = 1 + (1)(1) = 2
nueva w_{31} — anterior w_{31} + X3 y^ = 0 + (1)(1) = 1
los demás pesos se mantienen en cero.

Para el tercer par de vectores de entrenamiento e : s, (0 1 0 1):(0 1)
xj = 0, x_2 = 1, X3 = 0, X4 = 1
Yl = 0, y_2 = 1
nueva $w_2$2 — anterior $w_2$2 + X2 Y2 = 0 + (1)(1) = 1
nueva W42 — anterior W42 + X4 y_2 = 0 + (1)(1) = 1
los demás pesos se mantienen en cero.

Para el último par de vectores de entrenamiento e : s, (0 0 0 1):(0 1)

los demás pesos se mantienen en cero.

Por lo tanto, la matriz de pesos queda de la siguiente forma:

$$\mathbf{W} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 2 \end{bmatrix}$$

El listado 9.5 muestra un programa en VHDL para entrenar una red heteroasociativa. El programa lee los patrones de entrada de un archivo *patron.in* y escribe la matriz de ponderaciones en el archivo *pesos.out.*

```
1 USE std.textio.all;
2
3 ENTITY heteroasociativa IS END;
4
5 ARCHITECTURE red OF heteroasociativa IS
6    FILE archivo_entrada: TEXT OPEN READ_MODE IS "patron.in";
7    FILE archiVO_salida: TEXT OPEN WRITE_MODE IS "pesos.out";
8    TYPE ponderaciones IS ARRAY (0 TO 3, 0 TO 1) OF INTEGER;
9    TYPE entrada IS ARRAY (0 TO 3) OF INTEGER;
10   TYPE salida IS ARRAY (0 TO 1) OF INTEGER;
11 BEGIN
12   PROCESS
13     VARIABLE buf_in, buf_out: LINE;
14     VARIABLE w : ponderaciones : = ((0,0),(0,0), (0,0), (0,0));
15     VARIABLE x : entrada;
16     VARIABLE y : salida;

18  BEGIN
19      WHILE not endfile(archivo_entrada) LOOP
20         readline(archivo_entrada, buf_in);
21         FOR i IN 0 TO 3 LOOP
22            read(buf_in,x(i));
23         END LOOP;
24         read(buf_in,y(0));
```

Cotinúa

```
25          read(buf_in,y(1));
26          FOR i IN 0 TO 3 LOOP
27             FOR j IN 0 TO 1 LOOP
28                w(i,j): =  w(i,j) + x(i)*y(j);
29             END LOOP;
30          END LOOP;
31      END LOOP;
32      FOR i IN 0 TO 3 LOOP
33         FOR j IN 0 TO 1 LOOP
34            write(buf_out, w(i,j));
35            write(buf_out, " ");
36         END LOOP;
37         writeline(archivo_salida,buf_out);
38      END LOOP;
39      WAIT;
40   END PROCESS;
41 END red;
```

Listado 9.5 Programa en VHDL para entrenar una red heteroasociativa.

El listado anterior es un programa interesante, ya que se muestra por primera vez el uso de arreglos en una y dos dimensiones en VHDL.

Para declarar un arreglo en VHDL es necesario definir un tipo de dato de tipo arreglo mediante la palabra reservada TYPE.

La forma de declarar un arreglo en una dimensión es:

TYPE nombre_arreglo IS ARRAY (No.l TO No.2) OF tipo_dato;

El arreglo es una representación en un lenguaje de alto nivel como VHDL de un vector. En este caso se declara un vector con nombre nom*bre_arreglo* con dimensiones del No. *1* al No. 2. Los datos que puede almacenar son los declarados en *tipo dato.*

Un arreglo de dos dimensiones se declara de la siguiente forma:

TYPE nombre_arreglo IS ARRAY (No.l TO No.2,No.3 TO No.4) OF tipo_dato;

En este caso el arreglo tiene dos dimensiones, una de tamaño del No. 1 al No. 2 y la otra del No. 3 al No. 4. El arreglo en dos dimensiones es una representación de una matriz.

Una vez definido el tipo de dato, se debe usar para declarar el arreglo. En esta declaración se puede utilizar VARIABLE, CONSTANT o SIGNAL.

En las líneas 8, 9 y 10 se definen tres tipos de arreglos:

```
8   TYPE ponderaciones IS ARRAY (0 TO 3, 0 TO 1) OF INTEGER;
9   TYPE entrada IS ARRAY (0 TO 3) OF INTEGER;
10  TYPE salida IS ARRAY (0 TO 1) OF INTEGER;
```

En la línea 8 se define un arreglo para representar la matriz de ponderaciones (pesos) que tendrá las dimensiones de cuatro renglones por dos columnas; además, podrá almacenar enteros.

La línea 9 define un arreglo para los patrones de entrada, el cual tiene dimensiones de cuatro localidades. Por último, en la línea 10 se define un vector de dos localidades para representar la salida de la red.

En las líneas 14, 15 y 16 se declaran los arreglos.

```
14    VARIABLE w : ponderaciones : =  ((0,0),(0,0),(0,0),(0,0));
15    VARIABLE x : entrada;
16    VARIABLE y : salida;
```

La línea 14 declara una variable w que es del tipo ponderaciones, el cual a su vez está definido como un arreglo de dos dimensiones. Es importante observar en la línea 14 que también se están inicializando los valores para la matriz w.

En la línea 14 y 15 se declaran los arreglos para las entradas x y las salidas *y*.

A fin de leer los patrones de entrenamiento del archivo de entrada, se usa la instrucción FOR. Esta se ejecuta cuatro veces y en cada ocasión se lee una de las x de entrada.

```
20          readline(archivo_entrada, buf_in);
21          FOR i IN 0 TO 3 LOOP
22             read(buf_in,x(i));
23          END LOOP;
24          read(buf_in,y(0));
25          read(buf_in,y(1));
```

La línea 22 muestra cómo accesar una localidad del arreglo. Esto se realiza anotando el número de localidad entre paréntesis *(x(i));* es decir, la primera vez que se ejecute el FOR, se almacenará lo que se lea del archivo de entrada en la primera localidad del arreglo x. La segunda vez que se ejecute el ciclo se almacenará en *x(l),* la siguiente vez en x(2), etcétera.

Al terminar de ejecutarse el FOR, se leen los dos datos para el vector de salida *y*.

Para finalizar, se calculan los valores de la matriz de pesos con ayuda de dos ciclos anidados, como se muestra a continuación:

```
26          FOR i IN 0 TO 3 LOOP
27             FOR j IN 0 TO 1 LOOP
28                w(i,j): =  w(i,j)  + x(i)*y(j);
29             END LOOP;
30          END LOOP;
```

La forma de tener acceso a una localidad de un arreglo de dos dimensiones se aprecia en la línea 28.

Uso de una red heteroasociativa

Algoritmo para utilizar la red heteroasociativa

```
Inicializar los pesos.
Para cada patrón de entrada hacer
{
      Hacer x igual al patrón de entrada;


      Calcular la entrada ponderada a cada una de las neuronas
```

$$y_i = \begin{cases} 1; & \text{si } I > 0 \\ 0; & \text{si } I = 0 \\ -1; & \text{si } I < 0 \end{cases}$$

Determinar la activación de las salidas

}

Ejemplo 9.8 Pruebe que el ejemplo 9.7 funciona para las entradas dadas en el problema. Utilice la función de activación o transferencia:

$$y_i = \begin{cases} 1; & \text{si } I > 0 \\ 0; & \text{si } I \leq 0 \end{cases}$$

Solución

Según el ejemplo 9.7, la matriz de ponderaciones es:

$$\mathbf{W} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 2 \end{bmatrix}$$

Para el primer patrón de entrada,
$\mathbf{x} = (1\ 0\ 0\ 0)$
$I_1 = x_1w_{11} + x_2w_{21} + x_3w_{31} + x_4w_{41} = (1)(2) + (0)(0) + (0)(1) + (0)(0) = 2$
$I_2 = x_1w_{12} + x_2w_{22} + x_3w_{32} + x_4w_{42} = (1)(0) + (0)(1) + (0)(0) + (0)(2) = 0$
Por lo tanto,
$y_1 = 1$, dado que $I > 0$
$y_2 = 0$, dado que $I = 0$
La respuesta es correcta.

Para el segundo patrón de entrada,
$\mathbf{x} = (1\ 0\ 1\ 0)$
$I_1 = x_1w_{11} + x_2w_{21} + x_3w_{31} + x_4w_{41} = (1)(2) + (0)(0) + (1)(1) + (0)(0) = 3$
$I_2 = x_1w_{12} + x_2w_{22} + x_3w_{32} + x_4w_{42} = (1)(0) + (0)(1) + (1)(0) + (0)(2) = 0$
Por lo tanto,
$y_1 = 1$, dado que $I > 0$
$y_2 = 0$, dado que $I = 0$
La respuesta es correcta.

Para el tercer patrón de entrada,
$\mathbf{x} = (0\ 1\ 0\ 1)$
$I_1 = x_1w_{11} + x_2w_{21} + x_3w_{31} + x_4w_{41} = (0)(2) + (1)(0) + (0)(1) + (1)(0) = 0$
$I_2 = x_1w_{12} + x_2w_{22} + x_3w_{32} + x_4w_{42} = (0)(0) + (1)(1) + (0)(0) + (1)(2) = 3$
Por lo tanto,
$y_1 = 0$, dado que $I = 0$
$y_2 = 1$, dado que $I > 0$
La respuesta es correcta.

Para el cuarto patrón de entrada,
$\mathbf{x} = (0\ 0\ 0\ 1)$
$I_1 = x_1w_{11} + x_2w_{21} + x_3w_{31} + x_4w_{41} = (0)(2) + (0)(0) + (0)(1) + (1)(0) = 0$
$I_2 = x_1w_{12} + x_2w_{22} + x_3w_{32} + x_4w_{42} = (0)(0) + (0)(1) + (0)(0) + (1)(2) = 2$
Por lo tanto,
$y_1 = 0$, dado que $I = 0$
$y_2 = 1$, dado que $I > 0$

La respuesta es correcta.

Ejemplo 9.9 Demuestre la red heteroasociativa del ejemplo 9.6 para una entrada similar al patrón de entrenamiento.

Solución

Pruebe el vector (0 0 1 0), el cual difiere del vector de entrenamiento (10 10) en un solo bit.

Sea,

La red asocia al vector (0 0 10) con el vector (1 0), a pesar de que el vector de entrada difiere de los entrenados.

Ejemplo 9.10 Demuestre la red heteroasociativa para una entrada completamente diferente a los patrones de entrenamiento.

Solución

Pruebe el vector (0 1 10).

Sea,

La red asocia el vector (0 110) con el vector (1 1), el cual no se encuentra en los patrones de entrenamiento. La red responde correctamente al no encontrar similitud con los patrones de entrenamiento; por lo tanto, contesta con un vector diferente.

El programa en el listado 6 muestra una forma de programar la red heteroasociativa para el problema anterior.

```
1 ENTITY heteroasociativa IS
2   PORT (x1, x2, x3, x4 : IN INTEGER RANGE 0 TO 1;
3         y1, y2 : OUT INTEGER RANGE 0 TO 1) ;
4 END;

6 ARCHITECTURE red OF heteroasociativa IS
7   TYPE ponderaciones IS ARRAY (0 TO 3, 0 TO 1) OF INTEGER;
8 BEGIN
```

Continúa

```
9     PROCESS (xl, x2, x3, x4)
10      VARIABLE w : ponderaciones : = ((2,0),(0, 1),(1,0),(0,2));
11      VARIABLE II, 12: INTEGER;
12    BEGIN
13       II := xl*w(0,0) + x2*w(1,0) + x3*w(2,0) + x4*w(3,0);
14       12 := xl*w(0, 1) + x2*w(1, 1) + x3*w(2, 1) + x4*w(3, 1);
15       IF I1>0 THEN
16          yl< = 1;
17       ELSE
18          yl< = 0;
19       END IF;
20       IF I2>0 THEN
21          y2< = 1;
22       ELSE
23       y2< = 0;
24      END IF;
25   END PROCESS;
26 END red;
```

Listado 9.6 Programa en VHDL para utilizar la red heteroasociativa.

9.4.2 Producto de vectores para asociación de patrones

Los pesos encontrados por la ley de Hebb se pueden obtener a través de productos de vectores.

Sean dos vectores

El producto entre e y s es el producto entre la matriz E = e^T y la matriz S = s.

$$\mathbf{ES} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} \begin{bmatrix} s_1 & s_2 & \cdots & s_m \end{bmatrix} = \begin{bmatrix} e_1 s_1 & e_1 s_2 & \cdots & e_1 s_m \\ e_2 s_1 & e_2 s_2 & & \\ \vdots & & \ddots & \vdots \\ e_n s_1 & & \cdots & e_n s_m \end{bmatrix}$$

La matriz ES almacena los pesos encontrados entre un patrón e y una salida s. Para hallar la matriz de ponderaciones (pesos), es necesario sumar las matrices de los patrones de entrenamiento.

Ejemplo 9.11 Utilice el producto de vectores para encontrar la matriz de pesos del ejemplo 9.7.

Solución

Para el primer vector de entrada
e = (1 0 0 0), s = (1 0)
el producto entre vectores es,

$$\begin{bmatrix}1\\0\\0\\0\end{bmatrix}\begin{bmatrix}1 & 0\end{bmatrix}=\begin{bmatrix}1 & 0\\0 & 0\\0 & 0\\0 & 0\end{bmatrix}$$

Para el segundo vector de entrada
e - (1 0 1 0), s — (10)
el producto entre vectores es,

$$\begin{bmatrix}1\\0\\1\\0\end{bmatrix}\begin{bmatrix}1 & 0\end{bmatrix}=\begin{bmatrix}1 & 0\\0 & 0\\1 & 0\\0 & 0\end{bmatrix}$$

Para el tercer vector de entrada
e = (0 1 0 1), s — (0 1)
el producto entre vectores es,

$$\begin{bmatrix}0\\1\\0\\1\end{bmatrix}\begin{bmatrix}0 & 1\end{bmatrix}=\begin{bmatrix}0 & 0\\0 & 1\\0 & 0\\0 & 1\end{bmatrix}$$

Para el tercer vector de entrada
e = (0 0 0 1), s - (0 1)
el producto entre vectores es,

$$\begin{bmatrix}0\\0\\0\\1\end{bmatrix}\begin{bmatrix}0 & 1\end{bmatrix}=\begin{bmatrix}0 & 0\\0 & 0\\0 & 0\\0 & 1\end{bmatrix}$$

La matriz de pesos se obtiene al sumar todas las matrices resultantes para cada patrón de entrada.

$$w=\begin{bmatrix}1 & 0\\0 & 0\\0 & 0\\0 & 0\end{bmatrix}+\begin{bmatrix}1 & 0\\0 & 0\\1 & 0\\0 & 0\end{bmatrix}+\begin{bmatrix}0 & 0\\0 & 1\\0 & 0\\0 & 1\end{bmatrix}+\begin{bmatrix}0 & 0\\0 & 0\\0 & 0\\0 & 1\end{bmatrix}=\begin{bmatrix}2 & 0\\0 & 0\\0 & 0\\0 & 2\end{bmatrix}$$

que es el mismo resultado obtenido a través de la ley de Hebb.

Aplicación de la red heteroasociativa en reconocimiento de caracteres

Las redes heteroasociativas se han aplicado con éxito en el reconocimiento de caracteres. Como ejemplo se puede utilizar la red para asociar dos pares de vectores. El vector x tiene 30 bits de entrada, y el vector y, 15 bits de salida.

```
..X..          .X.     xxxx.     XX.
.x.x.          x.x     X...X     x.x
.x.x.          xxx     X...X     xx.
xxxxx          x.x     xxxx.     x.x
X...X          x.x     X...X     xx.
X...X                  xxxx.
X...X
```

Un vector de entrada y salida se puede convertir en una señal binaria (0, 1) o bipolar (−1, 1). Por ejemplo, el vector de salida,

```
.x.
x.x
xxx
x.x
x.x
```

se puede expresar como un vector binario donde las x representan ceros y los puntos denotan unos:

(0 1 0 1 0 1 1 1 1 0 1 1 0 1)

Después de entrenar la red, se puede probar con entradas que tengan ruido para ver cómo las asocia con vectores correctos. Algunos ejemplos donde se introdujo ruido son los siguientes:

es un bit que toma el valor de 1 cuando debería ser 0.
o es un bit que toma el valor de 0 cuando debería ser 1.

Entrada	#.x.#		
	.x.x.	Salida	.x.
	.x.x#		x.x
	XXXXX		XXX
	x.#.x		x.x
	X...X		x.x
	X...X		
Entrada	..x.#		
	.x.x.	Salida	,x.
	.x.x#		x.x
	XXXXX		XXX
	x.#.x		x.x
	x..#x		x.x
	X...0		
Entrada	,.x#.		
	.x.x.	Salida	.x.
	#x.x#		x.x
	OXXXX		XXX
	X...X		x.x
	o..#x		x.x
	X...O		
Entrada	..x..		
	#x.o.	Salida	.x.
	.x.x#		x.x
	XXOXX		XXX
	x.#.x		x.x
	O...X		x.x
	x#..x		

Entrada	xxxx#		
	x...x	Salida	xx.
	x.#.x		x.x
	XXXX.		XXX
	X...X		x.x
	X..#X		XX.
	xxxx.		
Entrada	xxxx.		
	x...x	Salida	xx.
	x#..x		x.x
	XOXX.		XXX
	X...X		x.x
	X..#X		XX.
	xxox.		
Entrada	oxxx.		
	x...o	Salida	xx.
	X...X		x.x
	xxxx.		xxx
	x..#x		x.x
	X#..X		XX.
	oxxx#		
Entrada	xoxx.		
	x...o	Salida	xx.
	X...X		x.x
	ooxx.		xxx
	O...X		x.x
	X...X		XX.
	oxxx.		

Ejercicios

1. Entrenar una red hebbiana para reconocer una compuerta lógica OR. Use un programa en VHDL.
2. Entrenar una red hebbiana para reconocer una compuerta lógica XOR. Use un programa en VHDL.
3. Entrenar un perceptrón para reconocer una compuerta lógica OR.
4. Entrenar un perceptrón para reconocer una compuerta lógica XOR.
5. ¿Por qué no es posible programar una compuerta XOR con los tipos de neuronas anteriores.[7]
6. Elaborar un programa en VHDL para entrenar la adaline.
7. Redactar un programa en VHDL para utilizar la adaline.
8. Realizar un programa en VHDL para entrenar una red heteroasociativa que acepte patrones de entrada bipolares (-1, 1).
9. Redactar el programa en VHDL para usar la red heteroasociativa de la pregunta 1.
10. Elaborar un programa en VHDL para encontrar la matriz de pesos de una red heteroasociativa, utilizando el método de productos de vectores.
11. Encontrar la matriz de pesos para el problema de reconocimiento de caracteres. Debe reconocer las letras A y B y asociarlas con su respectivo patrón de salida. Use un programa en VHDL.

Bibliografía

Maureen Caudill, Charles Butler (1992). *Understanding Neural Networks. Computer explorations.* Volumen 1: Basic Networks. A Bradford Book. The MIT Press.

James A. Freeman, David M. Skapura (1991). *Redes neuronales. Algoritmos, aplicaciones y técnicas de programación.* Addison Wesley/Diaz de Santos.

R. Beale, T. Jackson (1990). *Neural Computing: An Introduction.* Institute of Physics Publishing.

Timothy Masters (1993). *Practical Neural Networks Recipes in C++.* Academic Press.

Sthephen T. Welstead (1994). *Neural Network and Fuzzy Logic Applications in C/C++.* Wiley Professional Computing.

Apéndice A

Herramientas de soporte para la programación en VHDL

Si bien ya mencionamos con anterioridad las ventajas que presenta el diseño de circuitos ASIC mediante dispositivos lógicos programables, conviene recordar que una de las más importantes es el bajo costo de los elementos requeridos para el desarrollo de estas aplicaciones. Estos elementos de soporte básico son: una computadora personal, un grabador de dispositivos lógicos programables y el software de aplicación.

A.1 WarpR4, el software de aplicación

WarpR4 es una herramienta para el diseño con lógica programable creada por Cypress Semiconductor, la cual procesa varios tipos de entrada de datos (captura esquemática, compilador estándar de VHDL y la combinación de ambos) haciéndola muy flexible y funcional.

En la actualidad es uno de los estándares más usados en la industria, ya que presenta la característica de optimizar los diseños con rapidez y precisión utilizando tan sólo una pequeña área del circuito; además, ofrece una interfaz gráfica (Galaxy) amigable con el usuario.

En la parte correspondiente al hardware, WarpR4 permite la grabación en diferentes familias de dispositivos lógicos programables; por ejemplo, PLD (22V10, 20V8 y 16V8), CPLD (de la serie Cypress FLASH370™), CPLD (de la familia MAX340™) y FPGA (de la familia FPGA-pASIC380™).

El sofware WarpR4 es gratuito para universidades y sólo se requiere solicitarlo a:

Cypress Semiconductor Corporation
3901 North First Street
San José, CA 95134
408 943 2600
http://www.cypress.com

Distribuidores en México

- Unique Technologies
  www.unique-technologies.com
- Integración de sistemas electrónicos
  Tcls. 26 82 71 03, 51 20 27 33 y 57 65 28 72

Para cualquier información adicional puede comunicarse con los autores (dagonzal@campus.cem.itesm.mx,jalcala@campus.cem.itesm.mx)

A. 1.1 Para iniciar WarpR4

A fin de usar la herramienta de trabajo WarpR4, hay que tenerla instalada. En la figura A. 1 se muestra esta condición y, como se observa, en este software se encuentra la interfaz gráfica (*Galaxy*), el simulador *(Nova)*, las notas técnicas *(Release Notes)y* la barra de herramientas *(Warp Toolbar).*

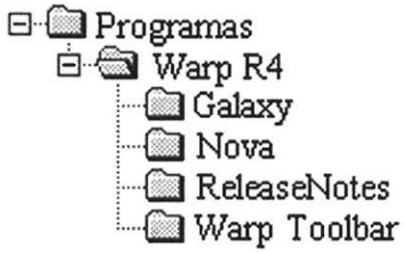


Figura A. 1 Software de soporte WarpR4-

A. 1.2 Galaxy: interfaz gráfica del usuario

Galaxy es la interfaz que permite la interacción entre el usuario y la herramienta de trabajo. En ella se realiza la edición, compilación y síntesis de los archivos escritos en código VHDL.

Para iniciar Galaxy es necesario entrar al menú de WarpR4 -y seleccionar Galaxy. Otra posibilidad es ejecutarla desde la barra de herramientas, la cual puede estar fija en la pantalla. La selección de Galaxy abre el menú gráfico mostrado en la figura A.2.

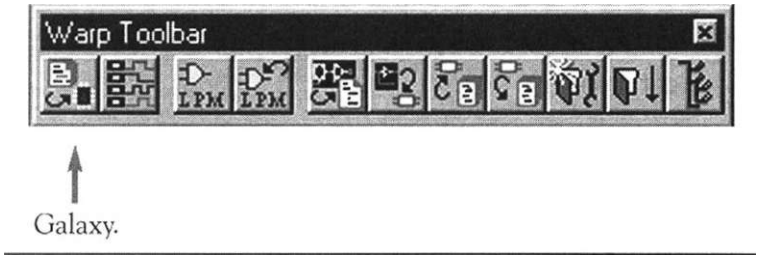


Figura A.2 Menú gráfico de Galaxy.

Creación inicial de un proyecto

Para iniciar un proyecto (Project) es necesario elegir la opción Project. Aquí se presentan dos casos: primero, al seleccionar Project por primera vez aparecerá una pantalla similar a la mostrada en la figura A.3. En ella el programa solicita la ruta en que se guardarán los proyectos (en nuestro caso: Ejemplo), así como el nombre que se asignará al archivo (diseños) en que se almacenarán los programas que generemos.

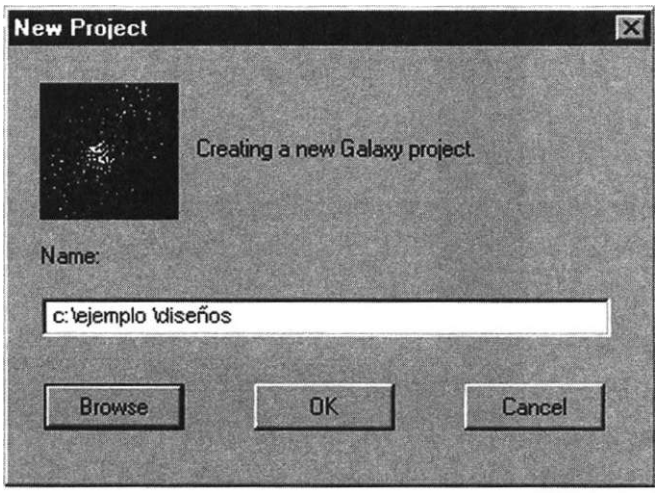


Figura A.3 Creación de un proyecto.

Segundo, si ya se tiene un proyecto, al seleccionar Galaxy aparecerá la pantalla de la figura A.4.

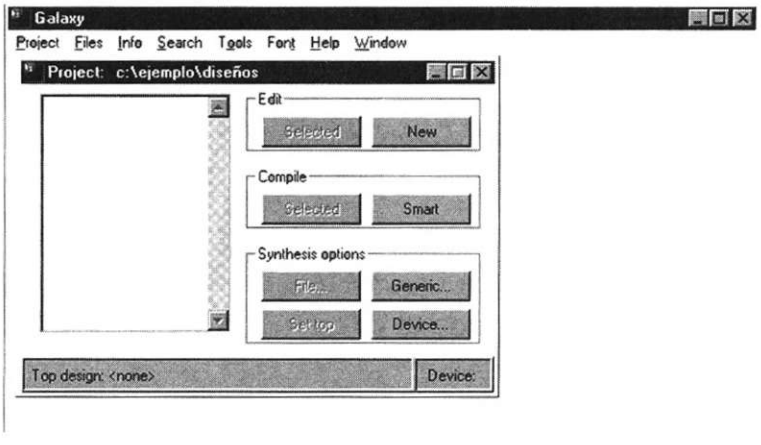


Figura A.4 Proyecto creado en Galaxy.

En la pantalla anterior podemos observar la presencia de un menú para Galaxy y otro para Project. En este último está el bloque Edit, que da la posibilidad de escoger un diseño almacenado (Select) o editar uno nuevo con la opción New. Con ésta aparecerá la pantalla de edición que se ilustra en la figura A.5.

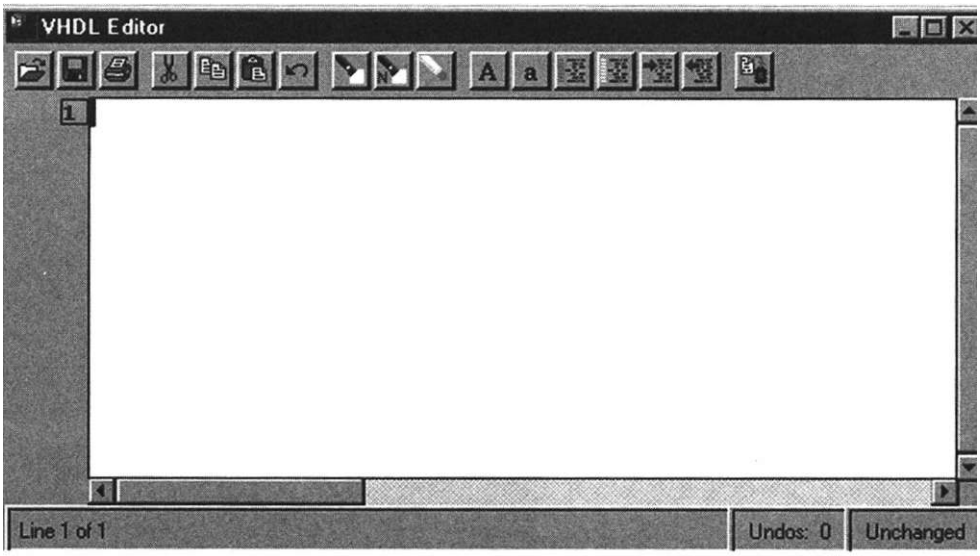


Figura A.5 Editor de Galaxy.

Para proseguir con el análisis, consideremos el listado del comparador de igualdad de 2 bits de la figura A.6.

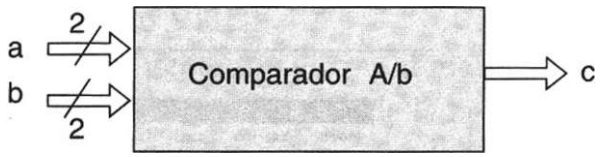


Figura A.6 Comparador de igualdad de 2 bits.

Ahora observemos la pantalla donde se edita el código VHDL, el cual corresponde al circuito ilustrado en la figura anterior (Fig. A.7).

Figura A.7 Edición de un programa.

Una vez que el programa ha sido editado, es necesario almacenarlo mediante la opción File->Save as, la cual se encuentra en el menú de Galaxy (Fig. A.4). Note cómo de forma automática el programa agrega la extensión .vhd (compara.vhd). El diseño así generado debe almacenarse para su uso posterior. Esta operación se realiza mediante el menú de Galaxy, comando File^Add. De esta manera el nuevo diseño aparecerá listado en la ventana de Project (Fig. A.8).

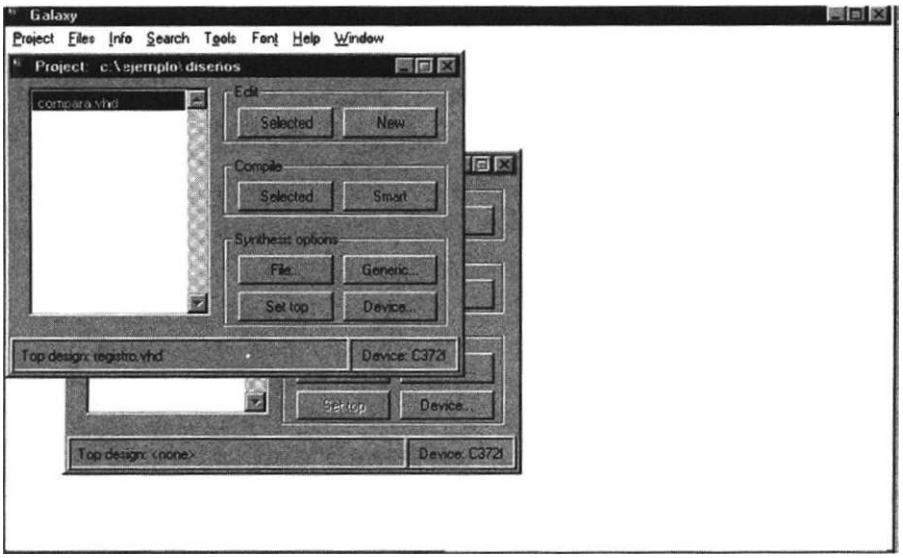


Figura A.8 Para añadir un diseño nuevo a un proyecto.

A.1.3 Compilación del diseño

La función de un compilador lògico es procesar y sintetizar el diseño, generando el archivo JEDEC (conocido como mapa de fusibles), el cual es reconocido por el grabador de dispositivos lógicos programables. Como una ventaja adicional, este tipo de compiladores detecta los errores de sintaxis y semántica, los cuales impiden la generación del archivo JEDEC.

Selección del dispositivo

Antes de compilar un diseño, es necesario elegir el dispositivo en que se grabará la aplicación. La selección se realiza en el bloque synthesis options del menú Project (Fig. A.8). En éste se escoge la opción dispositivo (device) y como resultado se abre la pantalla mostrada en la figura A.9.

En la parte superior izquierda de la pantalla se observan las opciones Device y Package. Entre las varias familias que ofrece Device, se elige la correspondiente al dispositivo que se va a utilizar (en nuestro caso, la familia C372I). En la opción Package se exhibe una lista de los dispositivos disponibles en la familia seleccionada; en ella se encuentra el tipo de encapsulado del dispositivo elegido, en este caso es el CY7C372I-66JC.

Otra de las opciones es la referente al voltaje de funcionamiento, el cual puede ser de 3.3 o 5 volts, según el dispositivo que se va a utilizar. En nuestro caso, el circuito CY7C372I funciona con 5 volts (consulte las hojas técnicas en el apéndice D).

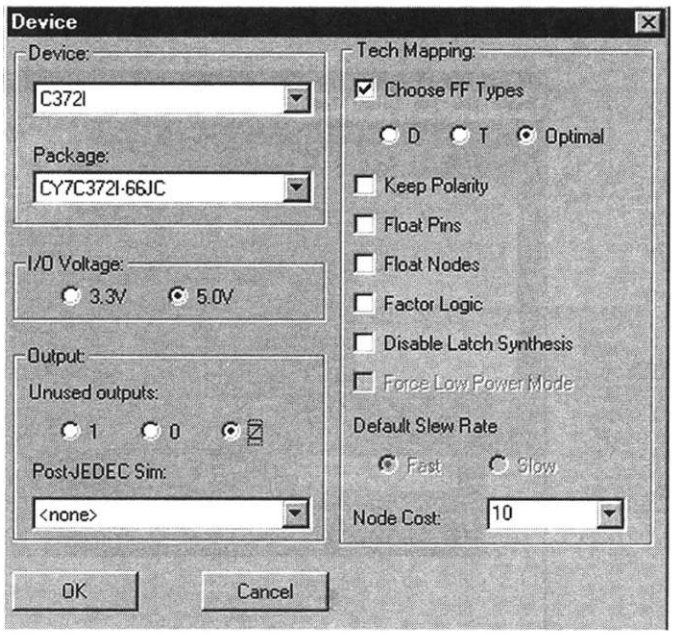


Figura A.9 Elección del dispositivo.

Tecnología del dispositivo

La tecnología del dispositivo (Tech Mapping) se refiere a las opciones que permiten la máxima optimización del circuito elegido, la cual se relaciona con la estructura de su arquitectura interna.

- *Selección de tipos de Flip'Flops*

Esta opción (Choose FF Types) se aplica en la síntesis de sistemas secuencia-Ies. Con ella, el diseñador puede escoger los diferentes tipos de flip-flops que se utilizan en la optimización de un diseño. La opción Opt es la recomendada, ya que permite al programa elegir de forma automática el flip-flop que mejor se adecúe a cada diseño.

- *Conservación de la polaridad*

Cuando esta opción (Keep Polarity) se habilita, el usuario tiene la posibilidad de elegir la polaridad deseada para cada salida. Cuando no está seleccionada, el programa asigna de manera automática la polaridad que considera más adecuada.

- *Terminales y nodos variables*

Durante el proceso de síntesis del programa, Warp genera diversas señales internas en el dispositivo, las cuales permiten realizar las conexiones necesarias para el óptimo funcionamiento del circuito. Estas señales conocidas como terminales *y* nodos variables (Float pins, Float nodes) se pueden ignorar al momento de implementar la lógica producida por los procesos de síntesis y optimización. Por lo general esta opción no se activa y estos cambios quedan sujetos al criterio del programador.

- *Factor lógico*

Es una opción (Factor Logic) aplicable únicamente a la familia MAX340 de la compañía Altera. Este módulo permite la implementación lógica en tres niveles en lugar del modo normal de dos niveles (suma de productos).

- *Nodos*

Nodos (Node cost), permite sintetizar el diseño de tal manera que sea compatible con otro. El valor recomendado es "10".

- *Opciones de simulación*

Esta opción (PostJEDEC Sim) sólo se aplica a PLD y CPLD. Consiste en un modelo de simulación que permite al usuario simular el diseño con información periódica). El menú PostJEDEC Sim exhibe una lista de simuladores disponibles. En nuestro ejemplo la opción no está seleccionada (<none>).

Opciones genéricas

En la opción Generics (Fig. A.10) se consideran diversas características relativas a la compilación.

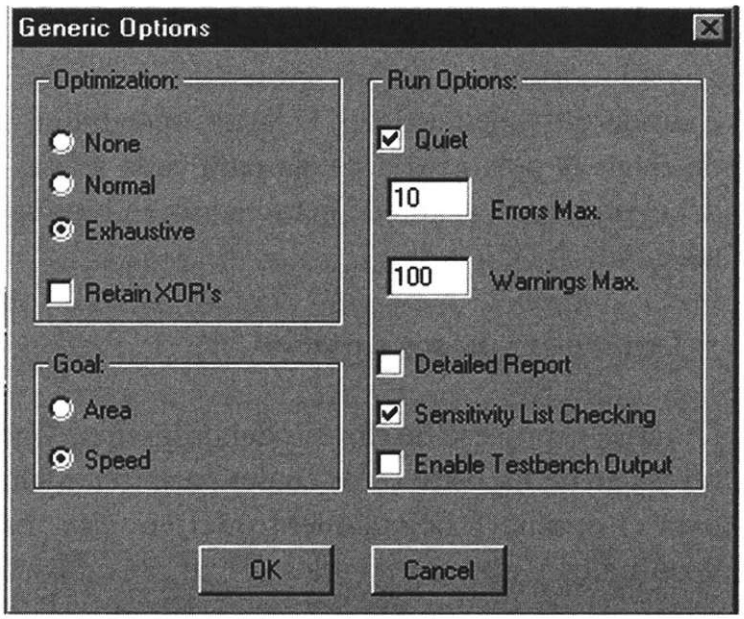


Figura A.10 Opciones genéricas.

Nivel de optimización

Esta propiedad (Optimization) se utiliza para indicar el nivel al que se quiere optimizar el diseño. El nivel exhaustivo permite una optimización que abarca diagramas de estado y condiciones no importa.

Retención de XOR

Esta opción (Retain XOR's) hace que Warp retenga los operadores XOR que se encuentren en el diseño con el fin de lograr una implementación más adecuada en el dispositivo.

• *Síntesis de campos*

Con esta opción (Goals) el usuario puede seleccionar el camino que seguirá la síntesis del diseño. Ahora bien, es importante hacer notar que sólo es recomendable para usuarios experimentados.

• Máximo *de errores y de condiciones de riesgo*

Ambas opciones (Max errors, Max Warnings) se utilizan principalmente cuando los diseños son largos. Su función es que el usuario especifique el número máximo de errores y condiciones de riesgo (warning) que se presentarán en la pantalla.

• *Reporte detallado*

Esta opción (Detailed Report) muestra un reporte detallado de la compilación de un archivo. No es recomendable habilitarla en diseños largos, debido a que el reporte que produce sería muy extenso.

• *Comprobación de la lista sensitiva*

Como su nombre lo indica, esta opción (Sensitivy List Checking) revisa las variables que forman la lista sensitiva de los procesos que se incluyen en los diseños. Si el programa no presenta procesos, esta opción no es recomendable.

Inicio de la compilación de un diseño

Antes de comenzar a compilar el diseño, es necesario que dentro del menú del proyecto se seleccione y enseguida la opción Set top, ya que ésta permite la compilación de dicho diseño (una vez realizado esto, el nombre del archivo debe aparecer resaltado en la pantalla).

Ahora bien, consideremos que hay dos caminos para iniciar la compilación:

1) Seleccionar de la ventana de proyecto el archivo que se desea compilar.

Se presiona Selected para iniciar la compilación.

2) La opción smart también permite compilar un diseño; pero se usa cuando se trata de programación *top level* (alto nivel) y hay que compilar varios archivos al mismo tiempo.

Una vez compilado el diseño (mediante la opción 1), se genera una pantalla que presenta, entre otros puntos, el resultado de los diversos procesos por los que atraviesa el diseño para su compilación adecuada (Fig. A.l 1).

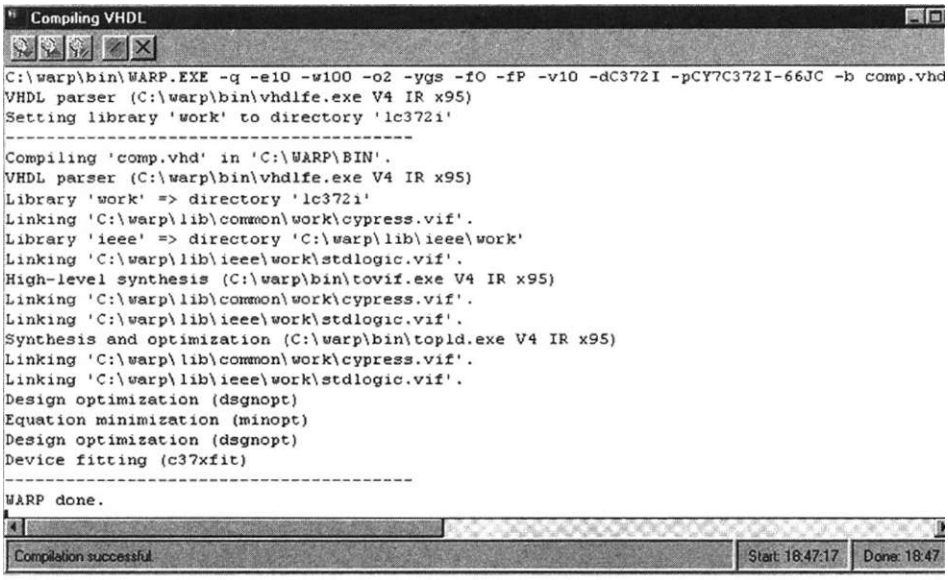


Figura A.l 1 Resultados de la compilación.

Si el diseño tiene errores, éstos aparecen en la ventana de compilación. La manera más sencilla de localizar un error es presionando el botón Lócate error , ya que esta opción indica en qué línea y columna se encuentra cada uno de los errores del código.

Una vez terminada la compilación se pueden guardar los mensajes en un archivo. Para esto tan sólo se escoge la opción Save as-»nombre.

A. 1.4 Nova

Nova (el simulador) es la herramienta que permite simular el comportamiento de un diseño basado en el trazado de formas de onda. La manera de entrar al simulador Nova es por medio de la barra de herramientas o desde el menú de Warp. La pantalla principal se muestra en la figura A. 12.

Dicha pantalla contiene un menú con las opciones File, Edit, Simula' te, Views y Options. Observe que en la sección izquierda de la pantalla hay una columna de botones en los cuales se indica el número de terminal (pin) y el número de nodos[1] asociados con las señales que se simularán (Fig. A. 12).

[1] Un nodo es un área del circuito que contiene uno o más puntos donde el usuario puede trazar una señal.

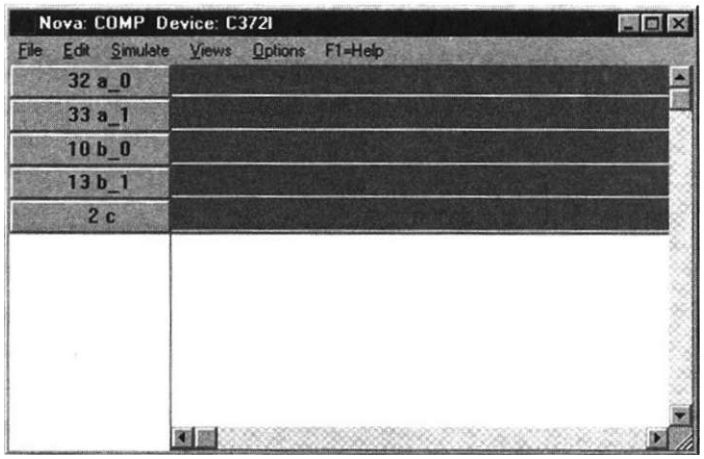


Figura A. 12 Simulador Nova.

Area de trazado

El área de trazado muestra los valores de las señales y nodos listados en la columna izquierda de la pantalla.

Para iniciar la simulación se debe asignar un valor a cada señal de entrada. Esto se realiza en la opción Edit, ya que contiene los diversos valores que puede tomar una señal (alto, bajo, señal de reloj y un pulso de reloj). Cada valor se asigna seleccionando primero el botón correspondiente a la señal y luego se indica un valor opcional (Fig. A. 13).

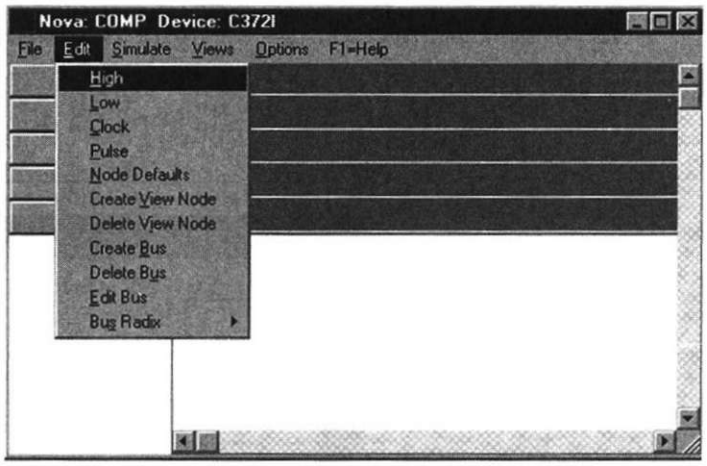


Figura A. 13 Asignación de valores a señales.

En nuestro ejemplo asignaremos el valor de "1" a las señales de entrada (a_0, a_1, b_0 y b_1).

A continuación se simula el diseño ejecutando el menú Simúlate-» Ejecute, el cual exhibe los resultados presentes en la figura A. 14.

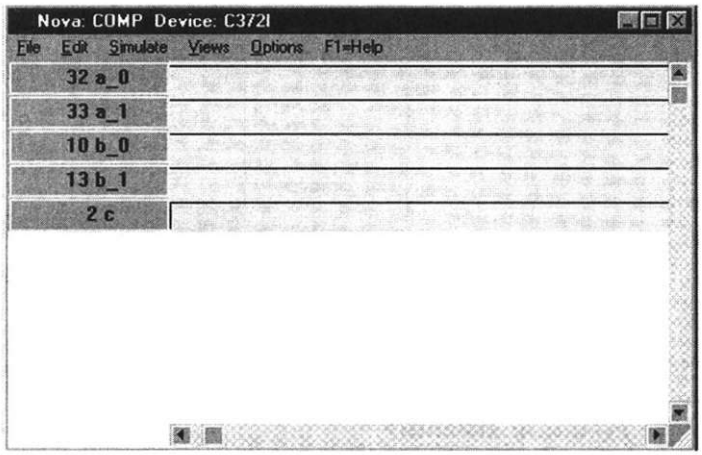


Figura A.14 Simulación de un archivo.

Note que la señal de salida c toma el valor de "1" porque los bits de entrada son iguales.

A.1.5 Archivo de reporte

La compilación del diseño trae consigo la generación de un archivo de reporte (extensión .rpt), el cual presenta información importante del diseño y de la forma en que se implementará en el dispositivo. Para analizar el archivo de reporte, se regresa a la pantalla principal y se elige el menú Info Report file, el cual abre una pantalla como la que se presenta en la figura A. 15.

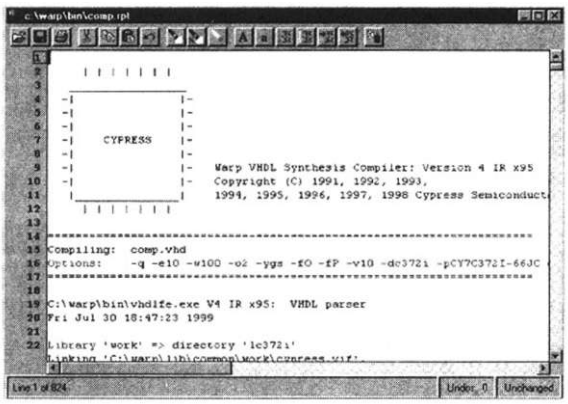
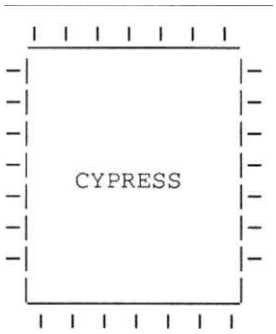

CYPRESS
Warp VHDL Synthesis Compiler: Version 4 IR x95
Copyright (C) 1991, 1992, 1993,
1994, 1995, 1996, 1997, 1998 Cypress Semiconduct
Compiling: comp.vhd
C:\warp\bin\vhdlfe.exe V4 IR x95: VHDL parser
Fri Jul 30 18:47:23 1999
Library 'work' => directory 'lc372i'

Figura A.15 Archivo de reporte.

Debido a que el archivo de reporte es muy extenso, expondremos brevemente cada una de las partes (Fig. A.16). Se recomienda que para mayor información se consulte el manual de usuario de Warp [2].



Información sobre los derechos reservados y la versión del programa

```
Warp VHDL Synthesis Compiler: Version 4 IR x95
Copyright (C) 1991, 1992, 1993,
1994, 1995, 1996, 1997, 1998 Cypress Semiconductor
```

Nombre del archivo y opciones activadas en la compilación

```
Compiling: comp.vhd
Options:-q -el0 -wl00 -o2 -ygs -f0 -fP -vl0 -dc372i -pCY7C372l-66JC comp.vhd

C:\warp\bin\vhdlfe.exe V4 IR x95:  VHDL parser
Fri Jul 30 18:47:23 1999
```

Fecha y tiempo de compilación

```
Library 'work' => directory 'lc372i'
Linking 'C:\warp\lib\common\work\cypress.vif'.
Library 'ieee' => directory 'C:\warp\lib\ieee\work'
Linking 'C:\warp\lib\ieee\work\stdlogic.vif'.
```

Referencias externas

```
C : \warp\bin\vhdl f e. exe : No errors.

C:\warp\bin\tovif.exe V4 IR x95:  High-level synthe-
sis
Fri Jul 30 18:47:26 1999

Linking 'C:\warp\lib\common\work\cypress.vif'.
Linking 'C:\warp\lib\ieee\work\stdlogic.vif'.
```

Ejecución del proceso de síntesis de alto nivel

```
C:\warp\bin\tovi f.exe:  No errors.
```

Figura A.16 Archivo de reporte.

En esta sección del reporte, conocida como *front end,* se muestran los datos generados por la herramienta VHDLFE, la cual revisa los errores de sintaxis y semántica que se pudieran presentar en el diseño. Otra de sus funciones es trasladar el archivo VHDL (por medio de simples ecuaciones y registros) a la siguiente fase, conocida como síntesis y optimización.

En la figura A. 17 se ilustra la parte del reporte generada en la sección de *optimización.* En términos generales se presentan las ecuaciones que rigen el comportamiento del diseño. La notación utilizada para representar dichas ecuaciones es del tipo suma de productos (la / representa una negación, el * una función AND, el + una función OR). Cada ecuación impresa en este archivo espera ser introducida en una macrocelda de un bloque lógico en particular.

```
PLD Optimizer Software: DSGNOPT.EXE    11/NOV/97    [v4.02 ]  4 IR x95

    OPTIMIZATION OPTIONS         (18:47:38)

    Messages :
      Information: Optimizing Banked Preset/Reset requirements.
    Summary:

                      Error Count = 0      Warning Count = 0

    Completed Successfully

              /a_1 + /a_0 + /b_1 + /b_0
            + a_1  + /a_0 + b_1  + /b_0
            + /a_1 + a_0  + /b_1 + b_0
            + a 1  + a 0  + b 1  + b 0      <-- Ecuaciones de diseño

    Completed Successfully
```

Figura A.17 Proceso de optimización del diseño.

Una vez obtenidas las ecuaciones, el paso siguiente se denomina *Fitting; o* sea, la forma de implementar en el dispositivo la lógica producida por los procesos de síntesis y optimización. Por lo general este término describe el proceso de asignar recursos en arquitecturas del tipo CPLD.

En la figura A. 18 se puede observar la realización de las ecuaciones producidas en la etapa anterior, las cuales se introducirán en las macroceldas correspondientes a uno de los bloques lógicos con que cuenta el dispositivo (el CPLD CY372I).

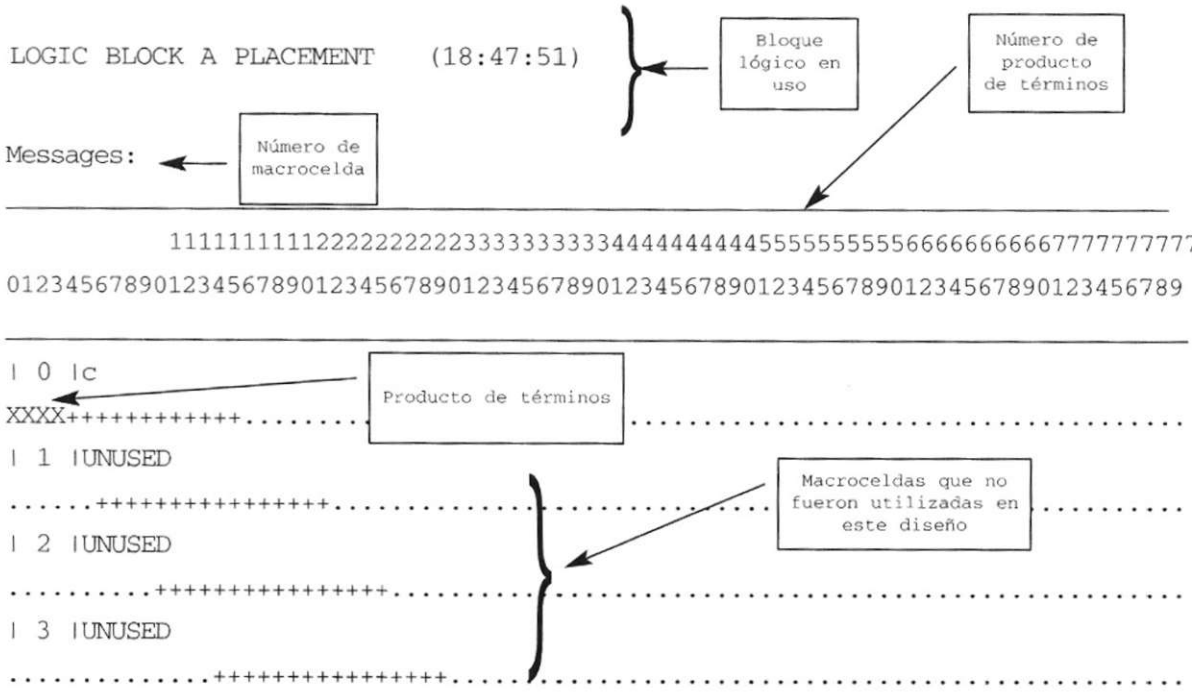


Figura A.18 Implementación de las ecuaciones en el bloque lógico.

En este bloque podemos observar que la primera columna indica el número de la macrocelda dentro del bloque lógico, donde la "X" representa un producto de términos (PT) y el signo + un espacio dentro del arreglo de PT que no está ocupado en esta macrocelda pero puede serlo en otra.

Otra de las partes incluidas en esta sección se refiere a las estadísticas, las cuales se incorporan porque presentan diversa información relativa al bloque lógico, como la cantidad de productos de términos realizados, las señales de control usadas, el porcentaje de utilización del bloque, etcétera (Fig. A.19).

```
Total count of outputs placed       =  1
Total count of unique Product Terms =  4
Total Product Terms to be assigned  =  4

Max Product Terms used / available  =  4 /  80   =5.1  %

Control Signals for Logic Block A
CLK pin   13  |  <not used>
CLK pin   35  |  <not used>
PRESET        |  <not used>
RESET         |  <not used>
OE 0          |  <not used>
OE 1          |  <not used>
OE 2          |  <not used>
OE 3          |  <not used>
```

Figura A.19 Estadísticas presentadas por bloque lógico.

Otra de las partes que incluye el archivo de reporte es la correspondiente a la distribución de terminales del dispositivo (Fig. A.20).

```
Device:  c372i
Package: CY7C372I-66JC
```

1	GND	23	GND
2	c	24	Not Used
3	Not Used	25	Not Used
4	Not Used	26	Not Used
5	Not Used	27	Not Used
6	Not Used	28	Not Used
7	Not Used	29	Not Used
8	Not Used	30	Not Used
9	Not Used	31	Not Used
10	b_0	32	a_0
11	VPP	33	a_l
12	GND	34	GND
13	b_l	35	Not Used
14	Not Used	36	Not Used
15	Not Used	37	Not Used
16	Not Used	38	Not Used
17	Not Used	39	Not Used
18	Not Used	40	Not Used
19	Not Used	41	Not Used
20	Not Used	42	Not Used
21	Not Used	43	Not Used
22	vcc	44	vcc

Figura A.20 Asignación de terminales para el CPLD C372i.

En el apéndice D se muestra la hoja técnica de este dispositivo. Recordemos que el programa asigna automáticamente las terminales (pines) con base en la distribución de productos de términos dentro de cada macrocelda.

Por último, se presenta la información correspondiente al mapa de fusibles (JEDEC), el cual se genera primero para cada bloque lógico y luego para el archivo de salida JEDEC (comp.jed), el cual finalmente se instala en el dispositivo elegido (Fig. A.21).

```
JEDEC ASSEMBLE                (18:47:52)

Messages :
   Information: Processing JEDEC for Logic Block 1.
   Information: Processing JEDEC for Logic Block 2.
   Information: Processing JEDEC for Logic Block 3.
   Information: Processing JEDEC for Logic Block 4.
   Information: JEDEC output file 'camp.jed' created.


Summary:

                    Error Count = 0      Warning Count = 0

Completed Successfully at 18:47:53
```

Figura A.21 Generación del archivo JEDEC.

A.1.6 Uso del programador ISR

Uno de los aspectos más importantes que hacen de un PLD un dispositivo dinámico es la capacidad de poder borrarlo y reprogramarlo sin necesidad de borradores ultravioleta. Algunos dispositivos lógicos programables se pueden borrar electrónicamente o mediante la opción Reprogramabilidad en sistema (ISR: In System Reprogrammable), donde la función se introduce desde la interfaz en paralelo de la computadora hacia el conector tipo serial JTAG. Este programador permite borrar, revisar y/o leer uno o varios dispositivos que empleen el estándar JTAG.

Características del programador ISR

El archivo que se utiliza para programar CPLD se llama isr.exe. Entre sus diversas funciones convierte el archivo con extensión .jed generado por Warp en el archivo con extensión .bit que se grabará en el CPLD. Para programar una aplicación se recomienda seguir los siguientes pasos:

1) Copiar el archivo isr.exe en el directorio elegido, por ejemplo[2]:

```
C: >md isr
C: > cd isr
C: \ISR> copy a: \isr.exe
```

2) Una vez que se tenga el archivo .jed generado en Warp, se usa el editor de MS-DOS para generar un archivo con extensión .dat. Por ejemplo,

```
C: \ISR>edit ejemplo.dat
```

3) En este archivo se deben de llenar los siguientes campos,

nombre_dispositivo[opción] [archivo.jed];

- El nombre_dispositivo se refiere al circuito integrado en que se quiere realizar la grabación.
- En el campo de opción se pueden declarar los siguientes parámetros:

p - para programar el CPLD con el archivo .JED
v - a fin de comprobar si lo que hay en el CPLD es el archivo .JED
r - para leer lo que hay en el CPLD y colocarlo en el archivo .JED
e - a fin de borrar lo que hay en el CPLD

Existen otras variantes que se pueden consultar en el manual de ISR [3].

- El campo archivo.jed es el lugar en que se declara el nombre del archivo (con extensión .jed) que se va a grabar.

La forma en que debe editarse dicho archivo se muestra en la figura A.22.

Figura A.2 Creación del archivo .dat.

2 Para lo cual se utilizan los comandos básicos de sistema operativo. Su uso en Windows no se especifica en este apartado.

4) Al terminar la edición, este archivo debe guardarse para poder continuar con la siguiente secuencia:

- Conectar el programador mediante su interfaz con el puerto paralelo de la PC.
- Conectar el programador a la base donde se encuentra el dispositivo.
- Polarizar el circuito CPLD con una fuente de 5 volts.

5) Ejecutar el siguiente comando en el símbolo del sistema:

```
C:/ISR/ISR/d ejemplo.dat
```

Si no hay problemas durante la grabación, el programa entregará un reporte en que indicará la ausencia de errores y que la aplicación se encuentra instalada en el dispositivo.

Bibliografía

David G. Maxinez. *Amplificación de Señales.* ITESM-CEM, 1993.

Cypress Semiconductor Corporation. *Applications Hand Book.* Cypress, Semiconductor Corp., 1994.

Cypress Semiconductor Corporation. *Programmable Logic Data Book.* Cypress, Semiconductor Corp., 1996.

Referencias

[1] Cypress Semiconductor Corp. *Software WarpR4.* 3901 First Street, San José California, CA 95134. Tel. (800)858-1810. Fax 408-943-68-48. Página Web www.cypress.com.

[2] Cypress Semiconductor Corporation. *The* User's *Guide.* Cypress, Semiconductor Corp., 1996.

[3] Cypress Semiconductor Corporation. An *Introduction To In System Reprogramming with FLASH370i.* Cypress Semiconductor Corp., 1996.

Apéndice B

Instalación del Software Warp

Warp se puede instalar en PC y plataforma Sun. Los requerimientos para cada uno se encuentran a continuación:

Requerimientos	Windows PC	Estaciones de trabajo Sun
Procesador	80486 mínimo	CPU SPARC
RAM	16 Mb mínimo	16Mb
Espacio en disco duro	60 Mb mínimo	60Mb
Sistema operativo	Windows 3.1 en adelante	Sun Os 4.1.1 y porteriores

Instalación en PC

Una vez que se han revisado los requerimientos anteriores, se siguen los siguientes pasos:

1) Cerrar todos las aplicaciones antes de correr el programa de instalación.
2) Insertar el CD y ejecutar el archivo pc\setup.exe (en el caso de una computadora personal).

Si se requiere tener acceso a la documentación en línea, es necesario instalar el programa Acrobat Reader, el cual se encuentra en ese CD-ROM. Para instalarlo existen dos formas:

a) Durante la instalación de Warp, aparece una pantalla con un mensaje que pregunta si se desea instalar Adobe Reader. Seleccione la opción Yes y el programa lo instalará.
b) Ejecute el archivo pc\acroread\ar32e30.exe y dé doble clic en este archivo.

Plataforma Sun

Para instalar Warp en Plataforma Sun (con sistema operativo SunOs 4.1.x/ Solaris 2.5 en adelante) o HP 9000 (serie 7000) se necesitan seguir los siguientes pasos:

SunOs 4.1.x

Una vez introducido el CD-ROM se ejecutan los siguientes comandos para crear el directorio */cdrom*
mkdir /cdrom

mount -rt hsfs /dev/srO /cdrom

En Solaris 2.5

Se ejecutan los siguientes comandos para crear el directorio /cdrom:
mkdir /cdrom

mount -F ufs -r /dev/dsk/c0t6d0s2 /cdrom

En HP-UX *10.10*

De igual forma que los anteriores, se ejecutan los siguientes comandos para crear el directorio /cdrom:
mkdir /cdrom
mount -o ro /dev/dsk/c0t2d0 /cdrom

Apéndice C

Identificadores, tipos y atributos

Al igual que otros lenguajes de alto nivel, VHDL utiliza diversos conceptos que son importantes al momento de realizar los programas, ya sea para facilitar la descripción del diseño o para simplificar las líneas de código utilizadas. Entre los conceptos más importantes se encuentran los identificadores, tipos de datos y atributos, los cuales se tratan en este apartado.

Identificadores

Los identificadores son los nombres o etiquetas que se usan para referir variables, constantes, señales, procesos, etc. Pueden ser números, letras y guiones bajos (_) que separen caracteres. Todos los identificadores deben respetar ciertas especificaciones o reglas para que se puedan compilar sin errores; por ejemplo, el primer carácter siempre es una letra, la cual puede ser minúscula o mayúscula (solo en el uso de identificadores).

VHDL cuenta con una lista de palabras reservadas (apéndice E) que no se pueden utilizar como identificadores porque son de uso exclusivo del compilador.

Objetos de datos

Un objeto en VHDL es un elemento del lenguaje que tiene un valor especifico; por ejemplo, un valor del tipo bit. En este lenguaje hay cuatro clases de objetos: *constantes, señales, archivos* y *variables.*

Constantes

Son objetos que mantienen siempre un valor fijo durante la ejecución del programa. De manera general, las constantes se usan para comprender

mejor el código, debido a que permiten identificar con más facilidad el valor que se les ha asignado.

La sintaxis para declarar una constante:

```
constant identificador: tipo : = expresión;
Ejemplos
constant Vcc: real : = 5.0;
constant cinco: integer := 3 4- 2;
constant tiempo: time := 100 ps;
constant valores: bit_vector := "10100011";
```

Como se puede observar, las constantes requieren un identificador (nombre), un tipo de datos y una expresión que indique el valor especifico que se le asigna.

Al momento de programar, las constantes son válidas sólo en la unidad de diseño en que se declaran; por ejemplo, una constante definida en una declaración de entidad es visible sólo dentro de la entidad; cuando la constante se declara en la arquitectura, nada más es visible en la arquitectura, y cuando se define en la región de declaraciones de un proceso, sólo es visible para ese proceso.

Señales

Son objetos utilizados como alambrados que permiten simular la interconexión de componentes dentro de la arquitectura de diseño. Estas señales permiten representar entradas o salidas de entidades que no tienen una terminal externa (patilla) al dispositivo.

Como ejemplo observemos el diagrama de la figura C1.

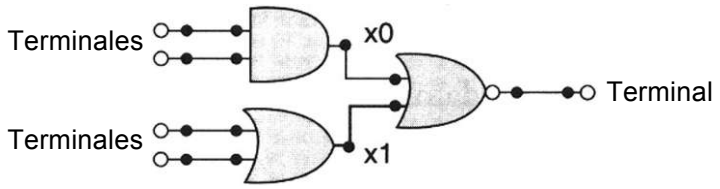


Figura C.l Ubicación de señales en un diseño.

Se puede apreciar que las señales etiquetadas como xO y xl no tienen asignada una terminal en la entidad de diseño, ya que sólo funcionan como un medio para interconectar las compuertas lógicas del circuito.

La forma de declarar una señal es la siguiente:
signal identificador: tipo: [rango}[1];
Ejemplos
signal vcc: bit: T;

signal suma: bit_vector (3 downto 0);

Variables

Una variable tiene asignado un valor que cambia continuamente dentro del programa. Estos objetos se utilizan para manejar datos aleatorios o que no tienen un valor especifico.

La forma en que se declara una variable es:

variable identificador (es): tipo[rango]: [expresión];
Ejemplos
variable contador: bit_vector (0 to 7);
variable x, y: integer;

Las variables no pueden proyectar formas de onda a su salida, debido a que su valor cambia constantemente, de modo que es imposible establecer un valor en cierto instante de tiempo.

Archivos

Un archivo es un objeto que permite la comunicación del diseño con su entorno exterior, ya que por medio de ellos se pueden leer y escribir datos cuando se hacen evaluaciones del circuito. Aquí cabe mencionar que un archivo es de un tipo de datos determinado y sólo puede almacenar datos de ese tipo.

La sintaxis para declarar un archivo es:

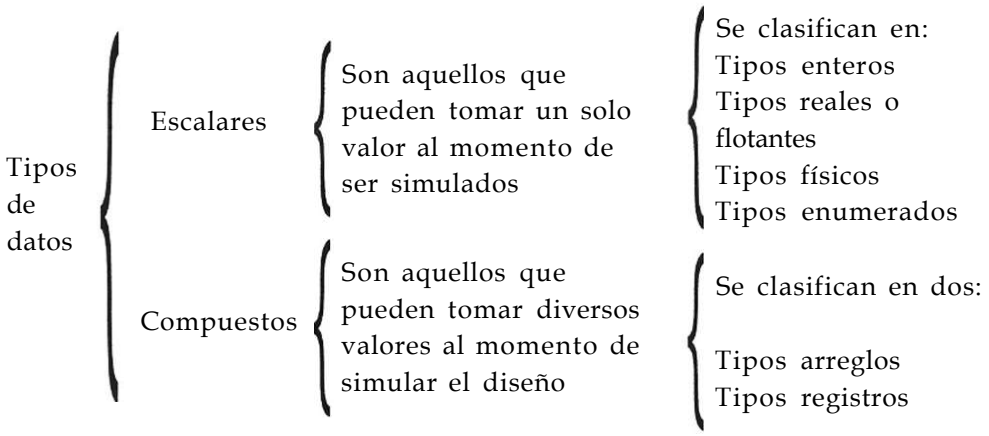
file identificador: tipo_archivo is [dirección "nombre";]
Ejemplos
file operaciones : Archivo_Enteros is in "datos.in";
file salidas : Archivo_Enteros is out "datos.out";

Tipos de datos

Un tipo de datos se utiliza para definir el valor que un objeto puede tomar, así como las operaciones que se realizan con ese objeto. En VHDL hay dos tipos básicos: compuesto y escalar, los cuales agrupan varios subtipos.

[1] Los parámetros indicados entre corchetes [ ), son valores opcionales.

A continuación se muestran estos tipos de datos y la forma en que se encuentran clasificados:


Tipos de datos
Escalares
Son aquellos que pueden tomar un solo valor al momento de ser simulados
Se clasifican en:
Tipos enteros
Tipos reales o flotantes
Tipos físicos
Tipos enumerados
Compuestos
Son aquellos que pueden tomar diversos valores al momento de simular el diseño
Se clasifican en dos:
Tipos arreglos
Tipos registros

a) Tipos escalares

En esta sección se describe a detalle los cuatro tipos escalares que existen en VHDL; también se muestran algunos ejemplos que permiten comprender mejor su uso.

Tipos enumerados

Este tipo se usa para listar los diversos valores que puede contener un objeto. Se llaman enumerados debido a que listan todos y cada uno de los valores que forman el tipo.

La sintaxis para declararlos es:
type identificador is definición_tipo;

Como se puede apreciar, la declaración del tipo contiene un nombre, el cual permite hacer referencia a él en el programa; también cuenta con el parámetro llamado definición del tipo, el cual corresponde a los valores que se le asignan.

Ejemplos
type nombres is (Ana, Mario, Julio, Cecilia);
type máquina is (edo_presente, edo_futuro, estado);
type letras is ('a\'b\ Y, *y\ V);
Los tipos bit y booleano se clasifican entre los tipos enumerados, debido a que pueden tomar más de un valor:
type booleano is (verdadero, falso);
type bit is ('0', T);

Tipos enteros y tipos reales

Como su nombre indica, los tipos enteros y reales sirven para representar números enteros y reales (fraccionarios), respectivamente. VHDL soporta valores enteros en el intervalo de -2,147-483,647 ($-2^{31}-1$) hasta 2,147,483,647 (2^{31}-1), y números reales en el intervalo de -1.0E38 a 1.0E38.

Ambos tipos —enteros y reales— siguen la misma sintaxis:

object identificador : type range [valores];

Un rango (range) es una palabra reservada por VHDL y se usa para definir un conjunto de valores. Cabe destacar que no todas las herramientas en VHDL manejan valores con signo. En nuestro caso el compilador utilizado (Warp) sólo maneja valores sin signo.

Tipos físicos

Se refiere a los valores que se usan como unidades de medida. En VHDL el único tipo físico que se encuentra predefinido es time (tiempo), el cual contiene como unidad primaria el *femtosegundo* (fs).

La manera de definir un tipo físico:

```
Type time is range 0 to 1E20
units
fs;
ps   =  1000 fs;
ns   =  1000 ps;
us   =  1000 ns;
ms   =  1000 us;
s    =  1000 ms;
min  =  60 sec;
h    = 6 0 min;
end units;
```

VHDL permite la creación de otros tipos físicos como metros, gramos, etc., sólo que en el diseño digital es difícil utilizar estos parámetros. Por esta razón nada más se muestran los tipos predefinidos, sin profundizar en el tema.

b) Tipos compuestos

Como ya se mencionó, los tipos compuestos pueden tener valores múltiples en un mismo tiempo al momento de ser simulados. Este tipo está formado por los arreglos y registros.

Tipo arreglo

El tipo arreglo está formado por múltiples elementos de un tipo en común. Estos arreglos se pueden considerar también como vectores, ya que agrupan varios elementos del mismo tipo.

La sintaxis utilizada para declarar un arreglo es:

```
type identificador is array (rango) of tipo_objetos;
```

Como se puede observar, en ambas declaraciones es necesario utilizar un valor (rango) que determine el conjunto de valores que va a tomar el tipo. En este caso el rango no se ha especificado, pero debe tomarse en cuenta que al momento de asignarlo se toma como un número entero positivo (número natural).

Los estándares IEEE 1076 y 1164 definen dos arreglos importantes, llamados bit_vector y std_logic_vector, los cuales forman parte de los tipos *bit* y *std_logic,* respectivamente. A continuación se puede observar la forma en que se declaran estos arreglos.

```
type bit_vector is array ( rango) of bit;
type std_logic_vector is array (rango) of std_logic;
```

El tipo std_logic es mas versátil que el tipo bit, debido a que incluye los valores de *alta impedancia* ('Z') y *no importa (-).*

Como ejemplo observemos las siguientes declaraciones de arreglos

```
type dígitos is array (9 downto 0) of integer;
type byte is array (7 downto 0) of bit;
type dirección is array (10 to 62) of bit;
```

Otro aspecto al utilizar arreglos es la facilidad que presentan para crear tablas de verdad:

```
type tabla is array (0 to 3, 0 to 2) of bit;
constant ejemplo: tabla := (
                    "00_0",
                    "01_0",
                    "10_0",
                    "11_1");
```

El arreglo declarado en este ejemplo es de dos dimensiones, ya que tiene un valor para el número binario que toman las entradas (de 0 a 3) y otro para el número de bits de entrada y salida (dos bits de entrada y uno de salida). Los guiones colocados entre los bits separan las entradas (lado izquierdo) de las salidas.

Tipo archivo (record)

A diferencia de los arreglos, los tipos archivo están formados por elementos de diferentes tipos, los cuales reciben el nombre de campos. Cada uno de estos campos debe tener un nombre que permita identificarlos con facilidad dentro del tipo.

Es importante destacar que el nombre de registro no tiene nada que ver con un registro en hardware utilizado para almacenar valores, ya que aunque los nombres son similares, en VHDL se toman como conceptos distintos.

La forma de declarar un tipo archivo es:

```
type identificador is record
Identificador : tipo;
end record;
```

Manejo de archivos en VHDL

Los archivos en VHDL son necesarios para leer información o almacenar archivos durante una simulación en VHDL. De hecho, su uso se restringe en la práctica en simulación.

Un archivo puede almacenar cualquier tipo de datos VHDL, aunque la declaración de archivos es diferente en VHDL87 y VHDL93, como se muestra a continuación:

```
TYPE ncnbre_tipo IS FILE OF integer;
FILE nombre: nanbre_tipo IS [modo] "archivo.txt";      —sólo VHDL'87
FILE nombre: nombre_tipo [OPEN modo] IS "archivo.txt"; —sólo VHDL'93
```

El modo en VHDL87 puede ser IN o OUT. Para VHDL93 es WRITE_MODE, READ MODE o APPEND_MODE.

Con objeto de leer información de un archivo se declara una variable de tipo LINE. En ella se almacena la información de toda una línea del archivo de texto. Cuando se quiera leer la información de la siguiente línea, se guarda en una variable del mismo tipo. Esta acción se realiza mediante la llamada función:

readline (nombre_archivo_salida, variable_tipo_line)

Después se debe leer información de la variable tipo LINE, utilizando la función READ.

read(variable_tipo_line, variable)

Una función muy útil al trabajar con archivos es: endfile(nombre). La cual regresa una variable booleana TRUE si se encuentra el fin de archivo y una FALSE en caso contrario.

El proceso de escritura es opuesto al proceso de lectura. Primero se debe mandar la información deseada a una variable tipo LINE mediante la siguiente función:

write(variable_tipo_line, variable)

Luego se escribe la línea al archivo de salida.

Writeline (nombre_archivo_salida, variable_tipo_line)

Las funciones para trabajar archivos se encuentran en el paquete textio, que se localiza en la librería std. Recuerde que la librería std se carga siempre que se compila un programa en VHDL.

Ejemplo: diseñar un programa para leer datos de un archivo y escribirlos en un archivo de salida en forma inversa.

El archivo de entrada puede tener

```
soccer          454345      5.5
basket          532466      6.7
hockey          847389      5.6
```

En el archivo de salida se genera

```
5.500000e+00  454345  soccer
6.700000e+00  532466  basket
5.600000e+00  847389  hockey
```

```
1 USE std.textio.all;

3 ENTITY text IS END;

5 ARCHITECTURE simple OF text IS
6     FILE archivo_entrada: TEXT OPEN READ_MODE IS "datos.in";
7     FILE archivo_salida: TEXT OPEN WRITE_MODE IS "datos.out";

9 BEGIN
10    PROCESS
```

```
11        VARIABLE buf_in, buf_0ut: LINE;
12        VARIABLE nombre: STRING(1 to 20);
13        VARIABLE num: INTEGER;
14        VARIABLE dato : REAL;
15    BEGIN
16        WHILE NOT endfile(archivo_entrada)
17        LOOP
18            readline (archivo_entrada, buf_in);
19            read (buf_in, nombre);
20            read (buf_in, num);
21            read (buf_in, dato);
22            write(buf_out,dato);
23            write(buf_out,"  ");
24            wri te(bu f_ou t,num);
25            write(buf_out,"  ") ;
26            wr i te(bu f_out,nombre);
27            writeline(archivo_salida,buf_out);
28        END LOOP;
29        WAIT;
30    END PROCESS;
31 END simple;
```

En la línea 6 se declara el archivo de lectura y en la línea 7 el de escritura. En la línea 11 se declaran dos variables de tipo LINE para mandar escribir y leer de ellos.

En las líneas 16 a 28 se encuentra un ciclo, el cual se ejecutará mientras no se encuentre el carácter de fin de archivo de *datos.in.*

La línea 18 lee una línea de archivo de entrada y la almacena en la variable *buf_in,* que es de tipo LINE. Posteriormente se leen diferentes datos de *buf_in* y se almacenan en las variables *nombre, num* y *dato.*

Por último se mandan escribir en forma inversa a la variable *buf_out* utilizando la función WRITE.

En la línea 28 se escribe en el archivo *datos.out* la información de *buf_out.*

Apéndice D

Hojas técnicas del CPLD Cy7C372i

Características del circuito

64 macroceldas distribuidas en cuatro bloques lógicos
32 terminales de entrada / salida
6 entradas dedicadas, incluyendo 2 terminales de reloj
Reprogramable en sistema (ISR[MR])
    Tecnología flash
    Interface JTAG
Alta velocidad
    F_{max} = 125 MHz
    T_{PD} = 10 ns
    T_s - 5.5 ns
    T_{co} = 6.5 ns
Totalmente compatible con PCI
Disponible en encapsulados PLCC de 44 terminales y CLCC
Compatibilidad en terminales con el CY7C37Ü

Descripción funcional

El circuito CY7C372i es un dispositivo lógico programable complejo (CPLD) reprogramable en sistema (ISR) y es parte de la familia FLASH370i[MR] de CPLD de alta funcionalidad y alta velocidad. Como todos los miembros de la familia FLASH370i, el CY7C372i está diseñado para brindar un fácil uso y alta funcionalidad.

Como todos los dispositivos de la familia FLASH370i, el CY7C372i es eléctricamente borrable y reprogramable en sistema (ISR[MR]), lo cual introduce en una misma arquitectura, tanto al circuito como al grabador, reduciendo

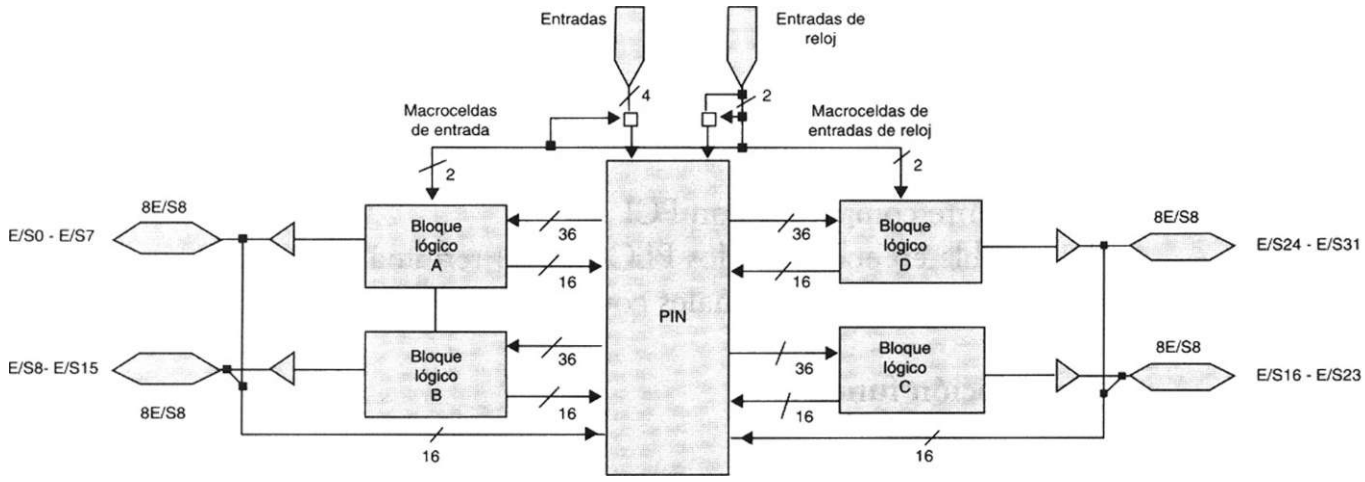
costos significativamente. La función ISR de Cypress, es implementada a través de 4 terminales de una interface serial. Los datos son desplazados y manejados (entrada y salida) a través de las terminales SDI y SDO respectivamente, usando la terminal del voltaje de programación (Vpp).

Las 64 macroceldas dentro del circuito CY7C372i están divididas en cuatro bloques lógicos. Cada bloque incluye 16 macroceldas, a 72 x 86 arreglos de productos de términos.

Los bloques lógicos dentro de la arquitectura FLASH370i, son conectados a través de un recurso extremadamente rápido llamado PIM (Matriz de Interconexión Programable).

Como todos los miembros de la familia FLASH370i, el CY7C372i es rico en recursos. Cada dos macroceldas en el dispositivo cuentan con terminales de E/S, resultando un total de 32 terminales de E/S del circuito. Además de cuatro entradas dedicadas y dos entradas de reloj.

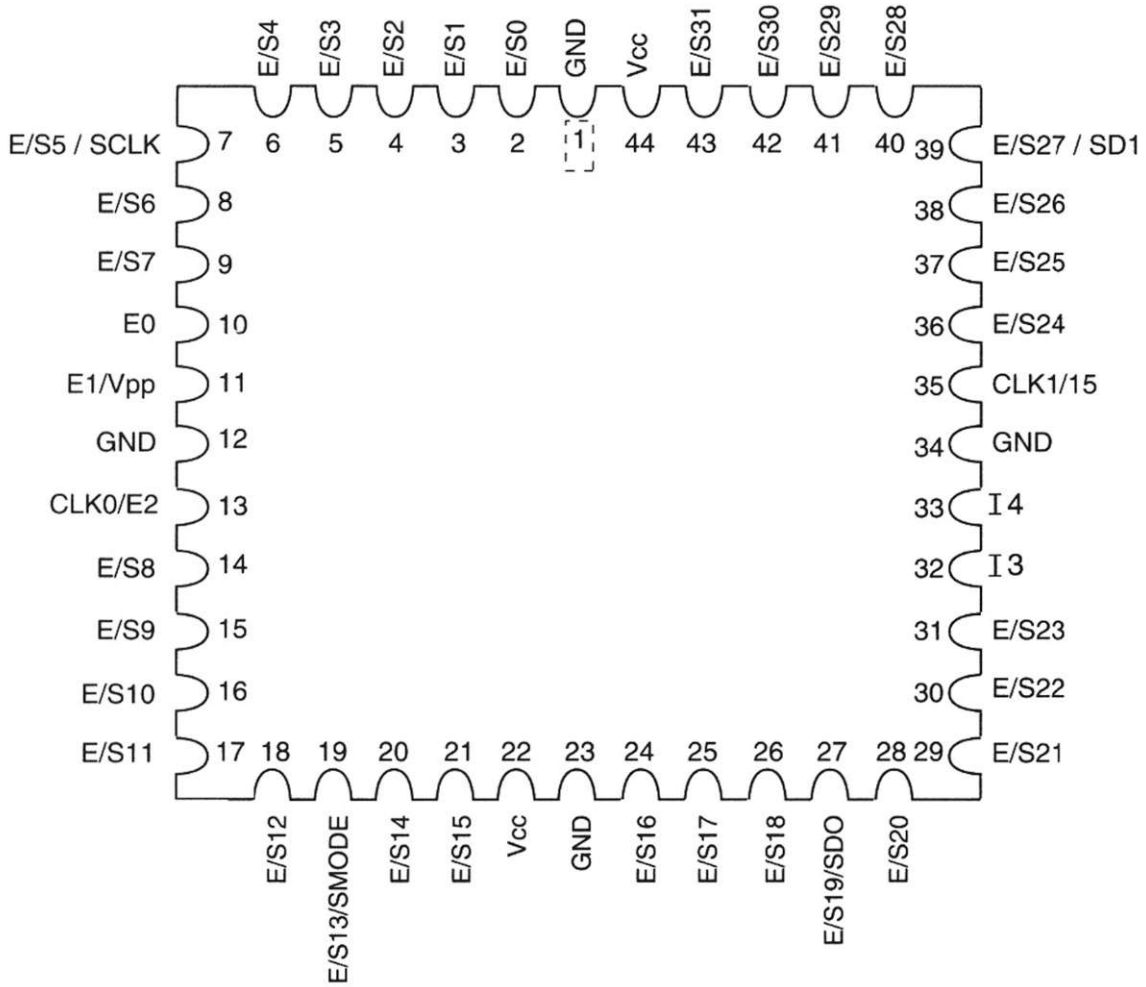
Finalmente, entre las características del CY7C372i se encuentra un sencillo modelo de tiempos. Como en otras arquitecturas de CPLD de alta funcionalidad, no presenta grandes retardos en la velocidad del circuito, así como efectos fanout, retardos de interconexión o retardos de expansión. Dependiendo del número de recursos utilizados o del tipo de aplicación, los parámetros de tiempo en el CY7C372i suelen ser los mismos.



Guía de selección

Características	7C372Í- 125	7C372Í - 100	7C372Í - 83	7C372i - 66	7C372i - 66
Máximo retardo de propagación tpp (ns)	10	12	15	20	20
Mínimo Set-up tS (ns)	5.5	6.0	8	10	10
Tiempo máximo de salida de la señal de reloj t^Q	6.5	6.5	8	10	10
Máxima Corriente entregada I_{cc} (mA) (comercial)	280	250	250	250	125

Configuración de terminales



Descripción funcional

Bloques Lógicos

El número de bloques lógicos distingue a cada miembro de la familia FLASH370i. El circuito CY7C372i incluye cuatro bloques lógicos. Cada bloque es construido de un arreglo de productos de términos y 16 macroceldas.

Arreglo de productos de términos

El arreglo de productos de términos en los bloques lógicos de la familia FLASH370i incluyen 36 entradas desde el PIM y 86 salidas de los productos de términos al producto de términos localizador. Las 36 entradas del PIM están disponibles en polaridades negativas y positivas, haciendo un arreglo de 72 **X** 86. Este arreglo tan largo en cada bloque lógico permite que una función compleja sea implementada en un sencillo paso dentro del dispositivo.

Producto de términos localizador

El producto de términos localizador, es un recurso dinámico y configurable que desplaza productos de términos a las macroceldas que los requieren. Cualquier número de productos de términos entre el 0 y 16 pueden ser asignados a cualquiera de las macroceldas contenidas en los bloques lógicos (esto es llamado ...). De aquí que los productos de términos puedan ser mezclados en múltiples macroceldas. Note que la implementación de los productos de términos en las macroceldas son realizadas por el software y son transparentes para el usuario.

Herramientas de desarrollo

El software de desarrollo para el CY7C372i está disponible en las versiones de Cypress Warp2MR, Warp2 +MR, y Warp3MR. Todos estos productos se basan en el lenguaje estándar VHDL. Cypress también cuenta con el soporte para otras herramientas tales como ABELMR, CUPLMR, y LOGIC MR.

Rangos Máximos de operación

Temperatura almacenada	-65 °C a + 150 °C
Temperatura ambiente con potencia aplicada	-55 °Ca +125 °C
Voltaje de funcionamiento o potencial de tierra	-0.5V a +7.0V
Voltaje DC de entrada	-0.5V a + 7.0V
Voltaje DC de programación	12.5V
Salida de corriente	16mA

Apéndice

Palabras reservadas en VHDL

A continuación se muestra una lista de las palabras reservadas en VHDL. Como se mencionó en el apéndice C, ninguna palabra reservada se puede usar como identificador de señales.

Abs	Exit	Not	Signal
Access	File	Null	Shared
After	For	Of	Sla
Alias	Function	On	Sll
All	Generate	Open	Sra
And	Generic	Or	Sri
Architecture	Group	Others	Subtype
Array	Guarded	Out	Then
Assert	If	Package	To
Atribute	Impure	Port	Transport
Begin	In	Postponed	Type
Block	Inertial	Procedure	Unaffected
Body	Inout	Process	Units
Buffer	Is	Pure	Until
Bus	Label	Range	Use
Case	Library	Record	Variable
Compoennt	Lindage	Register	Wait
Configuration	Literal	Reject	When
Constant	Loop	Rem	While
Disconnect	Map	Report	With
Downto	Mod	Return	Xnor
Else	Nand	Rol	Xor
Elsif	New	Ror	
End	Next	Select	
Entity	Nor	Severity	

La lista de palabras reservadas por el lenguaje se tomó del estándar IEEE Std 1076 -1993 del *Manual de Referencia del Lenguaje VHDL,* impreso por el Instituto de Ingenieros Eléctricos y Electrónicos en 1994.

Sección E2: Operadores definidos en VHDL según su orden de precedencia

Apéndice F

Operadores definidos en VHDL según su orden de precedencia

Operador	Descripción	Tipos de operandos	Resultado
**	potencia	Entero operador entero Real operador entero	Entero Real
Abs	Valor absoluto	Numérico	ídem operando
not	negación	Bit, booleano, vectores de bits	ídem operando
*	multiplicación	Entero operador entero Real op real Físico op real Físico op entero Entero op físico Real op físico	Entero Real Físico Físico Físico Físico
/	División	Entero op entero Real op real Físico op entero Físico op real Físico op físico	Entero Real Físico Físico Físico
Mod	módulo	Entero op entero	Entero
+	suma	Numérico op numérico	ídem operandos
-	resta	Numérico op numérico	ídem operandos
&	concatenación	Vector op vector Vector op elemento Elemento op vector Elemento op elemento	Vector Vector Vector Vector

Continúa

Operador	Descripción	Tipos de operandos	Resultado
sil	Desp. Lógico izquierdo	Vectores de bits op entero	Vector de bits
srl	Despl. Lógico derecho	Vector de bits op entero	Vector de bits
sla	Despl. Arit. izquierdo	Vector de bits op entero	Vector de bits
rol	Rotación izquierda	Vector de bits op entero	Vector de bits
ror	Rotación derecha	Vector de bits op entero	Vector de bits
	Igual que	No archivo op no archivo	Booleano
	Diferente que	No archivo op no archivo	Booleano
<	Menor que	No archivo op no archivo	Booleano
>	Mayor que	No archivo op no archivo	Booleano
< =	Menor o igual que	No archivo op no archivo	Booleano
> =	Mayor o igual que	No archivo op no archivo	Booleano
and	y lógica	Bit,booleano, bit_vector op bit, booleano, bit_vector	Idem operandos
or	o lógica	Bit,booleano, bit_vector op bit, booleano, bit_vector	Idem operandos
nand	y lógica negada	Bit,booleano, bit_vector op bit, booleano, bit_vector	ídem operandos
nor	o lógica negada	Bit,booleano, bit_vector op bit, booleano, bit_vector	Idem operandos
xor	or exclusiva	Bit,booleano, bit_vector op bit, booleano, bit_vector	Idem operandos
xnor	or exclusiva negada	Bit,booleano, bit_vector op bit, booleano, bit_vector	ídem operandos

Fuente de información: The IEEE Estándar VHDL Language Reference Manual, IEEE Std 1076-1987, 1988.

índice analítico

Archivos xxiv, 339-341
Arquitectura *(architecture),* 46-55
descripción
estructural, 53
funcional, 47-49
por flujo de datos, 49-52
estructural, 53
Arreglo (s)
(array), 298-299
de compuertas programables, 3-4,15-18,23
lógico genérico (GAL), 4,9-19
Asignaciones dobles, 87
Buffer tri-estado, 74
Carta ASM
descripción, 157
diseño mediante VHDL, 166-173
en comparación con máquina de estado, 159
estructura, 156
Circuito AMD 2909
descripción, 199-200
diseño y programación de componentes, 201-208
Codificadores, 87-88
Comparador de magnitud, 70-71
Compilación de un diseño, 316-319
Compiladores lógicos, 20
Componente *(component),* 53,201,206,262-264
Configuración *(configuration,* 37
Contadores, 101-104
con reset y carga en paralelo, 103
Controladores, 153-154
algoritmos de, 165
diseño, 162-166
CPLD *véase* dispositivos lógicos programables complejos
secuenciales, 46, 50, 69-75
Decodificadores, 83-87
BCD a decimal, 83-85
BCD a display de siete segmentos, 85-87
Diseño jerárquico, 53,197-208, 261-270
Metodología de diseño, 198
Dispositivos lógicos programables, 2-8,13
Dispositivos lógicos programables complejos, 2-4,13-15,23
Ecuaciones booleanas, 51
Entidad *(entity),* 37-46
declaración de, 40
diseño utilizando vectores, 42
diseño utilizando librerías y paquetes, 44
integración de, 123-127
Eventos *(event),* 96, 100
Flip-flop, 94-98
FPGA *véase* arreglos de compuertas programables
Full-custom, 3
GAL *véase* arreglo lógico genérico
Galaxy, 312-315
Identificadores, 42, 333
ISR programador, 327-329
Lenguaje de descripción en hardware
VDHL, 25-28,37
desventajas, 27-28
en la actualidad, 28
ventajas, 26-27
HDL, 25
Librerías
creación de, 208-210
declaración de, 45
ieee, 44
library, 44
work, 44
Lógica programable
ambiente de desarrollo, 18-19
campos de aplicación, 23-24
compañías de soporte, 28-31,32
futuro, 31
método tradicional de diseño, 20-22
Macroceldas, 12,14
Máquina de estado ASM, 154-159
bloque de estado, 154
bloque de decisión, 155
diseño con VHDL, 166

Microprocesadores, 230-31
programación de, 211-224,236-261
Modos, 39
buffer, 39
in, 39
inout, 39
out, 39
Multiplexores, 75
otros (*others),* 75
tipos lógicos estándar, 76
Netlist, 53
No importa (*dont care) véase* valores no importa,
Nova simulador, 320
Objetos de datos, 333-335
archivos, 335
constantes, 333
variables, 335
Operadores
aritméticos, 82
lógicos, 63
relaciónales, 73
Optimización, 323
PALASM, 21
Paquete (*package),* 37, 44, 122
cuerpo del, 37
declaración, 44
numeric_std, 46
numericjbit, 46
std_arith, 46,83
PLD *véase* dispositivos lógicos programables
Proceso *(process),* 48, 69
Programa de alto nivel (*top level),* 197, 200, 207
diseño del 198,223,264-266
Project, 313
Puertos, 39
Redes neuronales artificiales, 273-279
aprendizaje en las,
Redes asociativas, 294-297
Registros, 98-100
Reporte, archivo de, 322-327
Semi-custom, 3
Señal *(signal),* 54
Síntesis, 323
Sistema secuencial, 93-94
síncrono, 105
Sistema embebido, 229-237
clasificación, 235-236
diseño, 231-235
Sumadores 78
Tecnologías de fabricación
de circuitos integrados, 3
Tipo de datos, 36, 39,40
archivo, (record),
arreglo, 338
bit, 40
bitjvector, 40, 42, 43
boolean, 40
compuestos, 336
enumerado, 108, 336
escalar, 336
físicos, 337
iriteger, 40
reales, 337
Unidades de diseño, 37
primarias, 37
secundarias, 37
Valores no importa *(dont care),* 97
Variables, 335
VHDL *véase* lenguaje de descripción en hardware
WarpR4, 311-312
Instalación, 331-332

Esta obra se terminó de imprimir en abril del 2002
en los talleres de Litogràfica INGRAMEX, S.A. de C.V.
Centeno No. 162 Local 1, Col. Granjas Esmeralda
C.P. 09810, México, D.F.